POLITECHNIKA WARSZAWSKA

Rozprawa doktorska

Inżyniera Lądowa, Geodezja i Transport
Dziedzina nauk inżynieryjno-technicznych

mgr inż.
Mikołaj Miecznikowski

Zastosowanie sztucznej inteligencji w modelowaniu relacji konstytutywnych hipersprężystości

Promotor
dr hab. inż. **Marcin Gajewski**, prof. PW

Warszawa 2026

Podziękowania

Przede wszystkim pragnę złożyć najserdeczniejsze podziękowania mojemu promotorowi, Panu dr hab. inż. Marcinowi Gajewskiemu, prof. PW, za cotygodniowe spotkania na przestrzeni wielu lat, pełne merytorycznych dyskusji, które ukształtowały tę pracę, a także za wsparcie na duchu w trudniejszych momentach. Cenię sobie, że nasza współpraca miała charakter prawdziwie ludzki, daleki od czysto formalnej znajomości.

Słowa wdzięczności kieruję do Pana prof. dr hab. inż. Stanisława Jemioło za jego dorobek naukowy, którego uporządkowany sposób prezentowania zagadnień znacznie ułatwił mi przygotowanie tej rozprawy oraz za końcowe poprawki merytoryczne, które pozwoliły nadać jej ostateczny kształt.

Dziękuję zespołowi firmy ALS za możliwość pracy w siedzibie do późnych godzin wieczornych i „przepalenia odrobiny prądu" na trenowanie modeli, a także byłemu prezesowi, dr Sławomirowi Strzykowskiemu, za wprowadzenie mnie do świata sztucznej inteligencji oraz za słowa motywacji, które okazały się bezcenne.

Dziękuję mojemu przyjacielowi Przemysławowi za wyciąganie mnie zza biurka i nieustające próby utrzymania mnie w dobrej kondycji fizycznej, choć efekty bywały umiarkowane.

Dziękuję również mojemu przyjacielowi Jakubowi za rozmowy i rady w trudnych chwilach oraz za to, że w odpowiednim momencie potrafił przypomnieć, że życie toczy się też poza doktoratem.

Dziękuję mojej Siostrze za niezawodną pomoc i pewność, że w każdej potrzebie mogę na nią liczyć.

Wyrażam wdzięczność Mamie i Tacie za wychowanie, wiarę we mnie i codzienną troskę, także tę najbardziej namacalną, przy rodzinnym stole.

Na koniec szczególne podziękowania składam mojej narzeczonej Natalii za wyrozumiałość, nieustające wsparcie, motywację i rozmowy, dzięki którym mimo wszystko nigdy nie czułem się samotny.

Streszczenie

Rozprawa dotyczy modelowania konstytutywnego materiałów hipersprężystych z wykorzystaniem metod uczenia maszynowego uzupełnionych o więzy mechaniki ośrodków ciągłych. Mimo że praca mieści się w obszarze inżynierii lądowej, łączy ona zagadnienia sztucznej inteligencji z klasyczną mechaniką ośrodków ciągłych. Przyjęto w niej tezę, że wprowadzenie więzów fizycznych do modeli uczenia maszynowego jest warunkiem koniecznym poprawnego — fizycznie dopuszczalnego i zdolnego do ekstrapolacji — odtwarzania relacji konstytutywnych, a modele wzbogacone o te więzy pozwalają zarówno odkrywać te relacje, jak i rozwiązywać zagadnienia brzegowe oraz identyfikować parametry materiałowe wyłącznie na podstawie danych przemieszczeniowych i sił reakcji podporowych.

Punktem wyjścia jest studium klasycznych metod uczenia maszynowego. Spośród trzynastu przebadanych algorytmów żaden nie spełnił wymagań modelowania konstytutywnego bez wprowadzenia wiedzy kierunkowej. Wykazano przy tym, że typowe metryki (RMSE, R^2) nie uwzględniają spełnienia więzów fizycznych ani zdolności do ekstrapolacji. Stwierdzono, że do tego zadania najlepiej nadają się metody generujące ciągłe, interpretowalne funkcje regresji.

W części poświęconej odkrywaniu związków konstytutywnych przedstawiono identyfikację parametrów klasycznych modeli hipersprężystości oraz określanie nowych relacji z wykorzystaniem metod regresji liniowej i konstytutywnych sieci neuronowych (CANN). Opracowano moduł regresji umożliwiający analizę pełnych, różnorodnych stanów deformacji oraz wprowadzono pojęcie macierzy naprężeniowej cech. Wykazano konieczność uwzględnienia drugiego niezmiennika deformacji izochorycznej dla zestawu danych eksperymentalnych Treloara, a także pokazano, że regresja i CANN prowadzą do odmiennych, lecz poprawnych modeli materiałowych.

Kolejna część rozprawy dotyczy rozwiązywania zagadnień brzegowych metodą sieci neuronowych informowanych fizyką (PINN). Na przykładzie zagadnienia liniowej teorii sprężystości wykazano równoważność implementacji w czystym środowisku PyTorch oraz z użyciem biblioteki DeepXDE, a także przeprowadzono szczegółową analizę zbieżności, obejmującą wpływ normalizacji zmiennych, architektury sieci, sposobu narzucania warunków brzegowych oraz strategii optymalizacji na dokładność i koszt obliczeniowy.

W rozdziale wieńczącym pracę podejście PINN rozszerzono na materiał hipersprężysty, rozwiązując zagadnienie brzegowe rozciąganej tarczy z otworem, a następnie identyfikując

parametry konstytutywne modelu Yeoha. W tym celu zaadaptowano opisane w literaturze podejście do identyfikacji, w którym parametry materiałowe wyznaczane są łącznie z wagami sieci na podstawie pól przemieszczeń oraz historii obciążenia, opracowując dla niego modułową, rozszerzalną implementację, możliwą do zastosowania dla innych modeli konstytutywnych i zagadnień brzegowych. Potwierdzono odporność podejścia na szum pomiarowy o poziomie 1%, co wskazuje na możliwość jego zastosowania do rzeczywistych danych eksperymentalnych, np. pochodzących z cyfrowej korelacji obrazu. Ponadto stwierdzono, że dla zagadnień prostych metoda PINN nie przewyższa metody elementów skończonych, a jej potencjał ujawnia się dopiero w zagadnieniach odwrotnych.

Całość rozprawy potwierdza, że metody uczenia maszynowego z uwzględnieniem założeń fizycznych stanowią wartościowe i spójne z prawami mechaniki narzędzie modelowania konstytutywnego oraz identyfikacji parametrów materiałowych.

Słowa kluczowe: sztuczna inteligencja, uczenie maszynowe, konstytutywne sieci neuronowe (CANN), sieci neuronowe informowane fizyką (PINN), modelowanie konstytutywne, hipersprężystość, mechanika ośrodków ciągłych, analiza odwrotna.

Summary

This dissertation concerns the constitutive modeling of hyperelastic materials using machine learning methods supplemented with the constraints of continuum mechanics. Although the work falls within the field of civil engineering, it combines topics of artificial intelligence with classical continuum mechanics. The thesis adopted herein is that incorporating physical constraints into machine learning models is a necessary condition for the correct — physically admissible and extrapolation-capable — reproduction of constitutive relations, and that such physically constrained models make it possible both to discover these relations and to solve boundary value problems, as well as to identify material parameters solely on the basis of displacement data and support reaction forces.

The starting point is a study of classical supervised machine learning methods. Of the thirteen algorithms examined, none met the requirements for constitutive modeling without introducing domain knowledge. Moreover, it was shown that the typical metrics (RMSE, R^2) neither account for satisfying physical constraints nor for extrapolation capability. It was found that methods generating continuous, interpretable regression functions are best suited to this task.

The part devoted to the discovery of constitutive relations presents the identification of parameters of classical hyperelastic models and the discovery of new relations using linear regression and constitutive artificial neural networks (CANN). A regression module was developed that enables the analysis of complete, diverse deformation states, and the concept of a stress feature matrix was introduced. The necessity of accounting for the second invariant of isochoric deformation was demonstrated, and it was shown that regression and CANN lead to different yet correct material models.

The next part of the dissertation concerns the solution of boundary-value problems using physics-informed neural networks (PINN). Using the example of a linear elasticity problem, the equivalence of implementations in pure PyTorch and with the DeepXDE library was demonstrated for Treolar's test data, and an in-depth convergence analysis was carried out, covering the influence of variable normalization, network architecture, the manner of imposing boundary conditions, and the optimization strategy on the accuracy and computational cost.

In the concluding chapter, the PINN approach was extended to a hyperelastic material by solving the boundary value problem of a plate with a hole under tension and then

identifying the constitutive parameters of the Yeoh model. To this end, an identification approach described in the literature — in which the material parameters are determined jointly with the network weights on the basis of displacement fields and loading history — was adapted, and a modular, extensible implementation was developed for it, applicable to other constitutive models and boundary value problems. The robustness of the approach to measurement noise at a level of 1% was confirmed, indicating the possibility of its application to real experimental data, e.g. obtained from digital image correlation. Moreover, it was found that for forward problems, the PINN method does not outperform the finite element method, and its potential is revealed only in inverse problems.

The dissertation as a whole confirms that physically constrained machine learning methods constitute a valuable tool, consistent with the laws of mechanics, for constitutive modeling and the identification of material parameters.

Keywords: artificial intelligence, machine learning, constitutive artificial neural networks (CANN), physics-informed neural networks (PINN), constitutive modeling, hyperelasticity, continuum mechanics, inverse analysis.

Spis treści

[1] PINNACLE – *Physics-Informed Neural Network with Adaptive Constitutive Law Engine.*

Spis ważniejszych oznaczeń

Mechanika Ośrodków Ciągłych

$\mathbf{F}$, $\mathbf{U}$, $\mathbf{V}$	tensor gradientu deformacji, odpowiednio prawy i lewy tensor wydłużenia,
$\mathbf{R}$	tensor ortogonalny obrotu,
$\mathbf{E}$	tensor odkształcenia Lagrange'a,
$\mathbf{x}$, $\mathbf{X}$	wektor położenia punktu w konfiguracji aktualnej i konfiguracji odniesienia,
$\mathbf{u}$	wektor przemieszczenia,
$\mathbf{p}$	wektor sił powierzchniowych,
$\mathbf{C}$, $\mathbf{c}$, $\mathbf{B}$	prawy i lewy tensor deformacji Cauchy'ego–Greena,
$\mathbf{n}$, $\mathbf{N}$	wektory normalne do brzegu w konfiguracji aktualnej i w konfiguracji odniesienia,
$\boldsymbol{\sigma}$, $\boldsymbol{\tau}$, $\mathbf{S}$, $\mathbf{T}$	tensory naprężenia: Cauchy'ego, Kirchhoffa, Pioli–Kirchhoffa pierwszego rodzaju i Pioli–Kirchhoffa drugiego rodzaju,
$\mathbf{f}$, $\mathbf{f}_0$	wektor sił objętościowych w konfiguracji aktualnej i w konfiguracji odniesienia,
ρ, ρ_0	gęstość w konfiguracji aktualnej i w konfiguracji odniesienia,
$W(\cdot)$	funkcja jednostkowej energii sprężystości, umieszczane symbole nad tym oznaczeniem oznaczają zmianę argumentów,
TeMPO	Teoria Małych Przemieszczeń i Odkształceń,
JES	Jednostkowa Energia Sprężystości,
MOC	Mechanika Ośrodków Ciągłych,
UT	jednoosiowe rozciąganie (ang. *uniaxial tensile*),
PS	czyste ścinanie (ang. *pure shear*),
BT	dwuosiowe rozciąganie (ang. *biaxial tensile*).

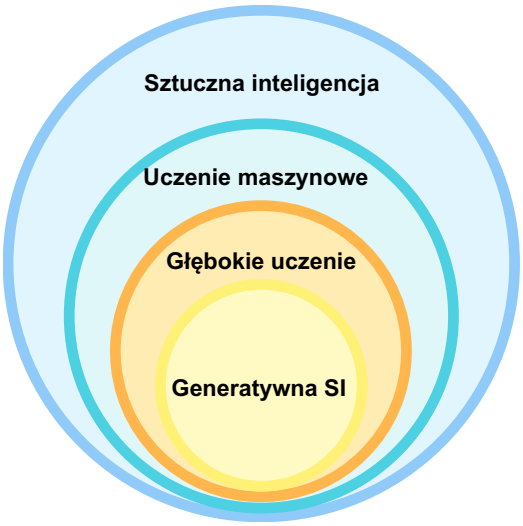
Sztuczna Inteligencja

L	funkcja kosztu,
L_1, L_2	normy regularyzacji Lasso i Ridge,
λ	parametr regularyzacji,
η	krok uczenia,

R^2	współczynnik determinacji,
MSE, RMSE	średni błąd kwadratowy i jego pierwiastek,
$\underline{W}$, $\underline{b}$	macierz wag i wektor obciążeń,
$\underline{\theta}$	wektor wszystkich parametrów sieci, zbiorczo wagi i obciążenia $\{\underline{W}^{(l)}, \underline{b}^{(l)}\}$,
σ	funkcja aktywacji,
FFN	jednokierunkowa sieć neuronowa (ang. *feedforward neural network*),
PINN	sieć neuronowa oparta na prawach fizyki (ang. *physics-informed neural network*),
CANN	konstytutywna sieć neuronowa (ang. *constitutive artificial neural network*).

1. Wstęp

W czasach, w których ilość danych rośnie w trybie wykładniczym, metody uczenia maszynowego (UM) stają się nieodzownym narzędziem do ich przetwarzania i analizy. Głównym atutem UM jest identyfikacja złożonych, często nieliniowych związków w zbiorach danych, których tradycyjne statystyczne metody nie są w stanie poprawnie opisać – szczególnie w inżynierii, ekonomii czy medycynie, gdzie związki fizyczne są zbyt zawiłe lub słabo odkryte, to algorytmy sztucznej inteligencji (SI) pozwalają na konstruowanie modeli predykcyjnych opartych wyłącznie na pomiarach. W trakcie ostatniej dekady głębokie sieci neuronowe (ang. *Deep Neural Networks*) znalazły zastosowanie w wielu dziedzinach, takich jak przetwarzanie języka naturalnego czy widzenie maszynowe. Pomimo znacznych sukcesów w tych obszarach, techniki te dopiero w ostatnich latach zaczęły być wykorzystywane w obliczeniach naukowych. W szczególności użycie sieci neuronowych do rozwiązywania równań różniczkowych cząstkowych (PDE) [2] dało impuls do rozwoju nowej dziedziny znanej jako *Scientific Machine Learning* (SciML) [3].



Rysunek 1.1. Hierarchia dziedzin w ramach sztucznej inteligencji [1]

Hierarchia dziedzin wchodzących w skład sztucznej inteligencji, pokazana na rys. 1.1, bywa często mylona — terminy takie jak sztuczna inteligencja, uczenie maszynowe, głębokie uczenie, w którego skład wchodzą sieci neuronowe, bywają traktowane jako synonimy i choć każdy z nich wchodzi w zakres poprzedniego, to zasadniczo dotyczą różnych obszarów. W celu uporządkowania nazewnictwa już na początku należy stwierdzić, że [1]:

Sztuczna Inteligencja (SI) jest dziedziną nauki, która skupia się na projektowaniu, ocenie i implementowaniu algorytmów i systemów komputerowych, które wykonują zadania wymagające pewnego rodzaju inteligentnego zachowania przypominającego ludzkie - są to wnioskowanie, przetwarzanie języka naturalnego, rozpoznawanie wzorców, podejmowanie decyzji itp.,

Uczenie Maszynowe (UM) to poddziedzina sztucznej inteligencji, która skupia się na tworzeniu modeli statystycznych i systemów komputerowych, które uczą się na podstawie dużych zbiorów danych lub wchodzą w interakcje ze środowiskiem. Systemy te potrafią podejmować decyzje bez potrzeby manualnego zaprogramowania reguł. W uczeniu maszynowym istnieją trzy główne paradygmaty:

- uczenie nadzorowane, w którym modele uczą się na oznaczonych parach wejście-wyjście,
- uczenie nienadzorowane, w którym algorytmy samoczynnie znajdują wzorce na podstawie nieoznaczonych danych,
- uczenie przez wzmacnianie, w którym agent dostosowuje swoje zachowanie podczas interakcji ze środowiskiem w oparciu o system nagród i kar.

Różnicą między sztuczną inteligencją a uczeniem maszynowym jest to, że SI obejmuje wszystkie techniki używane do rozwoju inteligentnych systemów, podczas gdy uczenie maszynowe skupia się na dostosowywaniu parametrów (wag) modeli podczas treningu. Sam dobór hiperparametrów tych modeli może być założony *a priori*, przeszukany siatkowo lub dokonany przy wykorzystaniu metod sztucznej inteligencji,

Głębokie Uczenie (Deep Learning) to poddziedzina uczenia maszynowego, która używa wielowarstwowych sieci neuronowych, których wagi dostrajane są w procesie propagacji wstecznej błędu oraz optymalizacji gradientowej. Teoretyczną podstawą ich stosowania jest twierdzenie o uniwersalnej aproksymacji [4], zgodnie z którym jednokierunkowa sieć neuronowa z co najmniej jedną warstwą ukrytą i odpowiednio dobraną funkcją aktywacji jest w stanie przybliżyć dowolną funkcję ciągłą z zadaną dokładnością, pod warunkiem dostatecznej liczby neuronów,

Generatywna Sztuczna Inteligencja to nowy kierunek badań w sieciach neuronowych skupiony na modelowaniu rozkładów w danych i generowaniu nowych próbek (np. tekstów, obrazów czy dźwięków), których cechy statystyczne są zbliżone do danych, na których zostały wytrenowane.

W szeroko rozumianej inżynierii lądowej zagadnienia sztucznej inteligencji rozwijane są od przełomu lat osiemdziesiątych i dziewięćdziesiątych XX wieku, kiedy to wzrost mocy obliczeniowej oraz upowszechnienie algorytmu propagacji wstecznej błędu umożliwiły praktyczne stosowanie sieci neuronowych w zadaniach inżynierskich [5], [6]. Pierwsze zastosowania dotyczą przede wszystkim analizy i projektowania konstrukcji, predykcji

właściwości materiałów (w tym wytrzymałości betonu), zagadnień geotechnicznych, detekcji uszkodzeń oraz wspomagania zarządzania przedsięwzięciami budowlanymi. Przeglądu dorobku pierwszej dekady rozwoju dziedziny dokonał Adeli [7]. Następnie, wraz z dojrzewaniem metody, sieci neuronowe zaczęto wykorzystywać nie tylko jako narzędzie regresji danych pomiarowych, lecz również do bezpośredniego modelowania zachowania materiałów. Przełomowa w tym zakresie była praca Ghaboussiego i współpracowników [8], w której relację konstytutywną betonu w płaskim stanie naprężenia (PSN) odwzorowano siecią neuronową uczoną wprost na danych doświadczalnych, bez zakładania *a priori* postaci krzywej naprężenie–odkształcenie. Podejście to rozwinięto następnie w kierunku integracji sieci neuronowych z metodą elementów skończonych (MES) [9] w charakterze przyrostowych modeli konstytutywnych [10], [11]. Monograficznego ujęcia metod algebry komputerowej oraz inteligencji obliczeniowej w zagadnieniach mechaniki konstrukcji dostarcza opracowanie [12].

Współcześnie obserwuje się ponowny, intensywny wzrost zainteresowania metodami uczenia maszynowego w Mechanice Ośrodków Ciągłych (MOC), napędzany rozwojem głębokiego uczenia oraz wspomnianej dziedziny SciML. Szczególne znaczenie mają przy tym podejścia, w których wiedza fizyczna nie jest pomijana, lecz wbudowywana w strukturę modelu lub w funkcję kosztu. Do tej grupy należą sieci neuronowe informowane fizyką (ang. *Physics-Informed Neural Networks*, PINN) [2], rozwiązujące zagadnienia brzegowe poprzez minimalizację residuum równań różniczkowych. W mechanice ciał stałych zastosowano je m.in. do zagadnień liniowej sprężystości, sprężysto-plastyczności oraz analizy odwrotnej [13]. Należą do niej również konstytutywne sieci neuronowe (ang. *Constitutive Artificial Neural Networks*, CANN) [14], [15], których architektura z założenia spełnia więzy termodynamiki i mechaniki ośrodków ciągłych, a ich wagom można nadać interpretację parametrów materiałowych. To właśnie na styku MOC oraz tak ujętych metod uczenia maszynowego sytuuje się tematyka niniejszej rozprawy.

Mimo że dysertacja osadzona jest w inżynierii lądowej, w jej tytule pojawia się termin „sztuczna inteligencja". Przyjęto w niej stanowisko, że choć SI stanowi fascynujący i dynamicznie rozwijający się kierunek badań, to z inżynierskiego punktu widzenia znacznie bardziej wartościowe są modele UM uwzględniające więzy fizyczne — np. prawa mechaniki ośrodków ciągłych oraz wytrzymałości materiałów. Należy jednak pamiętać, że zaawansowane modele UM należy stosować z ostrożnością — zawsze w połączeniu

z wiedzą inżynierską i krytycyzmem, a właściwe zrozumienie ich ograniczeń i założeń jest niezbędne dla bezpiecznego i efektywnego wykorzystania, gdyż minimalizuje ryzyko błędnych wniosków i decyzji.

2. Teza, cel i zakres rozprawy doktorskiej

Punktem wyjścia niniejszej rozprawy jest obserwacja, że klasyczne algorytmy uczenia maszynowego, stosowane bez wiedzy kierunkowej, nie są w stanie poprawnie odtworzyć relacji konstytutywnych materiałów hipersprężystych. Generowane przez nie modele nie spełniają więzów mechaniki ośrodków ciągłych i nie ekstrapolują poza zakres danych treningowych. Włączenie wiedzy fizycznej do struktury modelu lub do funkcji kosztu pozwala uzyskać modele zarazem dokładne, interpretowalne i zdolne do ekstrapolacji. Spostrzeżenie to wyznacza tezę, cel oraz zakres pracy.

2.1. Teza rozprawy

Wprowadzenie więzów mechaniki ośrodków ciągłych do modeli uczenia maszynowego jest warunkiem koniecznym poprawnego — fizycznie dopuszczalnego i zdolnego do ekstrapolacji — modelowania konstytutywnego materiałów hipersprężystych. Tak ufizycznione modele, obejmujące regresję z odpowiednio dobraną bazą funkcyjną, konstytutywne sieci neuronowe (CANN) oraz sieci neuronowe informowane fizyką (PINN), umożliwiają zarówno określanie relacji konstytutywnych, jak i rozwiązywanie zagadnień brzegowych oraz identyfikację parametrów materiałowych wyłącznie na podstawie danych przemieszczeniowych i sił reakcji podporowych.

2.2. Cel i zadania badawcze

Celem głównym rozprawy jest opracowanie oraz weryfikacja metod uczenia maszynowego z więzami fizycznymi, służących do modelowania konstytutywnego i rozwiązywania zagadnień brzegowych teorii hipersprężystości. Realizacji tego celu podporządkowano następujące zadania badawcze:

- ocenę przydatności klasycznych algorytmów uczenia maszynowego do modelowania konstytutywnego oraz wykazanie ich ograniczeń w kontekście spełniania więzów fizycznych i zdolności do ekstrapolacji,
- identyfikację parametrów znanych modeli hipersprężystości oraz określanie relacji konstytutywnych metodami regresji i konstytutywnych sieci neuronowych (CANN),
- sformułowanie i implementację sieci PINN dla zagadnienia brzegowego liniowej teorii sprężystości wraz z analizą wpływu normalizacji zmiennych oraz strategii treningu na zbieżność rozwiązania,

- rozszerzenie podejścia PINN na materiały hipersprężyste oraz identyfikację parametrów konstytutywnych w zagadnieniu odwrotnym wyłącznie na podstawie pól przemieszczeń i historii obciążenia.

2.3. Zakres i układ rozprawy

Rozprawa otwiera się przedstawieniem podstawowych metod uczenia maszynowego (rozdział 3) oraz wskazaniem tych z nich, które są właściwe do modelowania związków konstytutywnych hipersprężystości. Z uwagi na to, że znaczną część pracy poświęcono sieciom neuronowym, ich podstawowe pojęcia omówiono w odrębnym rozdziale 4. W rozdziale 5 przedstawiono teoretyczne podstawy mechaniki ośrodków ciągłych (MOC), niezbędne do nałożenia więzów fizycznych na architekturę sieci neuronowych.

Rozdział 6 stanowi praktyczną realizację rozdziałów poprzedzających — poświęcono go identyfikacji klasycznych parametrów modeli hipersprężystości, a także odkrywaniu nowych relacji konstytutywnych z wykorzystaniem metod regresji oraz konstytutywnych sieci neuronowych. Rozdział 7 stanowi wprowadzenie do sieci neuronowych informowanych fizyką (PINN) [2]. Zwieńczono go rozwiązaniem zagadnienia brzegowego belki wspornikowej w teorii małych przemieszczeń i odkształceń (TeMPO). Rozdziałem kończącym pracę (rozdział 8) jest rozwiązanie zagadnienia brzegowego teorii hipersprężystości dla rozciąganej tarczy z otworem oraz wyznaczenie parametrów konstytutywnych modelu Yeoha [16] wyłącznie na podstawie danych przemieszczeniowych i sił reakcji podporowych.

Układ rozprawy odzwierciedla logiczny ciąg postępowania: od danych doświadczalnych, poprzez określenie i identyfikację parametrów relacji konstytutywnych hipersprężystości (regresja, CANN), do rozwiązania zagadnień brzegowych metodą PINN — najpierw dla materiału liniowo sprężystego, a następnie dla materiału hipersprężystego, wraz z identyfikacją jego parametrów konstytutywnych.

3. Uczenie maszynowe

3.1. Eksploracyjna analiza danych

Celem niniejszego rozdziału jest zwrócenie uwagi na konieczność poprawnego przygotowania danych oraz znalezienie relacji między zmiennymi i zwrócenie uwagi na to, by na ten element pracy poświęcić odpowiednią ilość czasu. Ważną częścią wykonywania modeli uczenia maszynowego jest odpowiednie pozyskanie i przygotowanie danych. Przykładami popularnych serwisów umożliwiających bezpłatne pozyskiwanie danych są serwisy takie jak Kaggle [17], Google Dataset Search [18], UC Irvine's Machine Learning [19] czy DataHub [20].

Niezależnie od źródła pozyskania, zbiór danych przed dalszą analizą warto opisać formalnie. Przyjmijmy, że dysponujemy n parami $\{(\underline{x}^{(i)}, \underline{y}^{(i)}) : i = 1, \ldots, n\}$, gdzie $\underline{x}^{(i)} \in \mathbb{R}^d$ to wektory zmiennych objaśniających, a $\underline{y}^{(i)} \in \mathbb{R}^m$ to odpowiadające im wartości zmiennych objaśnianych. Macierze obserwacji $\underline{X} \in \mathbb{R}^{n \times d}$ oraz $\underline{Y} \in \mathbb{R}^{n \times m}$ zawierają odpowiednio wektory zmiennych objaśniających i objaśnianych:

$$\underline{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \ldots & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \ldots & x_d^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(n)} & x_2^{(n)} & \ldots & x_d^{(n)} \end{bmatrix}, \quad \underline{Y} = \begin{bmatrix} y_1^{(1)} & y_2^{(1)} & \ldots & y_m^{(1)} \\ y_1^{(2)} & y_2^{(2)} & \ldots & y_m^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ y_1^{(n)} & y_2^{(n)} & \ldots & y_m^{(n)} \end{bmatrix}. \tag{3.1}$$

3.2. Ocena zależności między zmiennymi

W eksploracyjnej analizie danych niezbędne jest zrozumienie relacji między zmiennymi. Istnieją dwie ważne metody statystyczne do ocenienia zależności między zmiennymi - są to kowariancja i korelacja. Obie metody opisują siłę i kierunek liniowej zależności między dwiema zmiennymi. Kowariancję (cov) między dwiema zmiennymi $(\underline{x}_i, \underline{x}_j)$ definiuje się jako:

$$\operatorname{cov}(\underline{x}_i, \underline{x}_j) = \frac{1}{n-1} \sum_{k=1}^{n} (x_i^{(k)} - \bar{x}_i)(x_j^{(k)} - \bar{x}_j) \in (-\infty, +\infty), \tag{3.2}$$

gdzie $\underline{x}_i \in \mathbb{R}^n$ oznacza wektor obserwacji i-tej zmiennej, a $\bar{x}_i$ jest średnią po jego składowych, a indeks k jest numerem obserwacji. Kowariancje mogą przyjmować wartości z zakresu $(-\infty, +\infty)$, co czyni je niewygodnymi do porównań. W celu ich normalizacji wprowadza się korelację Pearsona:

$$r(\underline{x}_i, \underline{x}_j) = \frac{\operatorname{cov}(\underline{x}_i, \underline{x}_j)}{S_{x_i} S_{x_j}} \in [-1; 1], \tag{3.3}$$

gdzie S_{x_i} oraz S_{x_j} to odchylenia standardowe zdefiniowane jako [21]:

$$S_{x_i} = \sqrt{\frac{1}{n-1} \sum_{k=1}^{n} \left(x_i^{(k)} - \bar{x}_i\right)^2}, \qquad S_{x_j} = \sqrt{\frac{1}{n-1} \sum_{k=1}^{n} \left(x_j^{(k)} - \bar{x}_j\right)^2}, \tag{3.4}$$

Do ocenienia zależności między wszystkimi zmiennymi w zbiorze danych wprowadza się macierz korelacji $\mathbf{r}$ [22]:

$$\mathbf{r} = \begin{bmatrix} r(\underline{x}_1, \underline{x}_1) & r(\underline{x}_1, \underline{x}_2) & \cdots & r(\underline{x}_1, \underline{x}_d) \\ r(\underline{x}_2, \underline{x}_1) & r(\underline{x}_2, \underline{x}_2) & \cdots & r(\underline{x}_2, \underline{x}_d) \\ \vdots & \vdots & \ddots & \vdots \\ r(\underline{x}_d, \underline{x}_1) & r(\underline{x}_d, \underline{x}_2) & \cdots & r(\underline{x}_d, \underline{x}_d) \end{bmatrix} \tag{3.5}$$

Każdy element w macierzy korelacji jest ograniczony do przedziału $[-1, 1]$, gdzie jeśli:

- $r(\underline{x}_i, \underline{x}_i) = 1$ oznacza to doskonałą korelację ze sobą samym,
- $r(\underline{x}_i, \underline{x}_j) \in [-1; 1]$ dla $i \neq j$ reprezentuje stopień i kierunek liniowej zależności między $\underline{x}_i$ i $\underline{x}_j$. Jeśli r jest bliski 1 lub -1 ($|r| > 0.7$), oznacza to silną dodatnią lub ujemną zależność liniową między zmiennymi.

W analogiczny sposób możemy sformułować macierz kowariancji.

3.3. Ocena jakości dopasowania modeli

Aby ocenić dopasowanie modelu, należy wygenerować przewidywane wartości na testowej macierzy obserwacji. Te przewidywane wartości, zebrane w macierzy $\underline{\hat{Y}}$, odpowiadają dopasowanym wartościom $\underline{\hat{y}}^{(i)} = f_{\underline{W},\underline{b}}\left(\underline{x}^{(i)}\right)$ dla $i = 1, \ldots, n$, gdzie $f_{\underline{W},\underline{b}} : \mathbb{R}^d \to \mathbb{R}^m$.

- współczynnik determinacji, R^2:

$$R^2(\underline{Y}, \hat{\underline{Y}}) = 1 - \frac{\sum_{i=1}^{n} \left\| \underline{y}^{(i)} - \hat{\underline{y}}^{(i)} \right\|_2^2}{\sum_{i=1}^{n} \left\| \underline{y}^{(i)} - \bar{\underline{y}} \right\|_2^2}, \tag{3.6}$$

- błąd średniokwadratowy, MSE:

$$\mathrm{MSE}(\underline{Y}, \hat{\underline{Y}}) = \frac{1}{nm} \sum_{i=1}^{n} \left\| \underline{y}^{(i)} - \hat{\underline{y}}^{(i)} \right\|_2^2, \tag{3.7}$$

  przy czym $R^2 \in (-\infty; 1]$, a wartość R^2 bliższa 1 wskazuje, że model lepiej wyjaśnia zmienność $\underline{Y}$.

- średnia z modułów różnic, MAE:

$$\mathrm{MAE}(\underline{Y}, \hat{\underline{Y}}) = \frac{1}{nm} \sum_{i=1}^{n} \left\| \underline{y}^{(i)} - \hat{\underline{y}}^{(i)} \right\|_1, \tag{3.8}$$

- mediana z modułów różnic, MedAE:

$$\mathrm{MedAE}(\underline{Y}, \hat{\underline{Y}}) = \operatorname*{Med}_{\substack{i=1,\ldots,n \\ j=1,\ldots,m}} \left| y_j^{(i)} - \hat{y}_j^{(i)} \right|, \tag{3.9}$$

- względny błąd w normie Manhattan, ϵ_{ℓ_1}:

$$\epsilon_{\ell_1}(\underline{Y}, \hat{\underline{Y}}) = \frac{\sum_{i=1}^{n} \left\| \underline{y}^{(i)} - \hat{\underline{y}}^{(i)} \right\|_1}{\sum_{i=1}^{n} \left\| \underline{y}^{(i)} \right\|_1}, \tag{3.10}$$

- względny błąd w normie euklidesowej, ϵ_{ℓ_2}:

$$\epsilon_{\ell_2}(\underline{Y}, \hat{\underline{Y}}) = \frac{\sqrt{\sum_{i=1}^{n} \left\| \underline{y}^{(i)} - \hat{\underline{y}}^{(i)} \right\|_2^2}}{\sqrt{\sum_{i=1}^{n} \left\| \underline{y}^{(i)} \right\|_2^2}}, \tag{3.11}$$

gdzie n to liczba próbek, m to liczba składowych wektora wyjściowego, a $\underline{y}^{(i)}$ oraz $\hat{\underline{y}}^{(i)} \in \mathbb{R}^m$ oznaczają odpowiednio rzeczywisty i przewidywany wektor wyjściowy i-tej próbki (czyli i-te wiersze macierzy $\underline{Y}$ oraz $\hat{\underline{Y}}$). Ich j-te składowe $y_j^{(i)}$ oraz $\hat{y}_j^{(i)}$ to rzeczywista i przewidywana j-ta wartość wyjściowa tej próbki. Wektor $\bar{\underline{y}} = \frac{1}{n} \sum_{i=1}^{n} \underline{y}^{(i)} \in \mathbb{R}^m$ to średni wektor wyjściowy po wszystkich próbkach, którego j-ta składowa $\bar{y}_j = \frac{1}{n} \sum_{i=1}^{n} y_j^{(i)}$ jest średnią j-tej składowej. Symbole $\|\cdot\|_1$ oraz $\|\cdot\|_2$ oznaczają odpowiednio normę Manhattan i normę euklidesową

wektora, tzn. dla wektora $\underline{v} = (v_1, \ldots, v_m) \in \mathbb{R}^m$ zachodzi $\|\underline{v}\|_1 = \sum_{j=1}^{m} |v_j|$ oraz $\|\underline{v}\|_2 = \sqrt{\sum_{j=1}^{m} v_j^2}$.

Poza popularnymi metrykami błędów w regresji liniowej stosuje się również bayesowskie kryterium informacyjne Schwarza, BIC [23] (ang. *Bayesian Information Criterion*) oraz kryterium informacyjne Akaikego, AIC [24] (ang. *Akaike Information Criterion*). Kryteria te są monitorowane podczas pracy, gdy chcemy ocenić zarówno jakość dopasowania modelu, jak i jego złożoność, co jest szczególnie istotne dla zachowania interpretowalności. Wzory dla kryteriów możemy zapisać:

- bayesowskie kryterium informacyjne Schwarza, BIC:

$$\mathrm{BIC}(\mathrm{MSE}_p, p, n) = n \log(\mathrm{MSE}_p) + p \log(n), \tag{3.12}$$

- kryterium informacyjne Akaikego, AIC:

$$\mathrm{AIC}(\mathrm{MSE}_p, p, n) = n \log(\mathrm{MSE}_p) + 2p, \tag{3.13}$$

gdzie MSE_p to błąd średniokwadratowy modelu o p parametrach, a n to liczba obserwacji w danych. Kryterium BIC stosuje większą karę za złożoność modelu, $p \log(n)$, co faworyzuje modele mniej złożone przy dużej liczbie danych. Kryterium AIC stosuje karę opartą wyłącznie na liczbie parametrów [21].

3.4. Wybrane metody regresyjne pod kątem wymagań mechaniki ośrodków ciągłych

Modelowanie zachowania mechanicznego materiałów metodami regresji zależy od wyboru odpowiedniego algorytmu uczenia maszynowego. Wybór ten wpływa na to, jak dobrze model dopasuje się do danych i jak dobrze będzie generalizować rozwiązanie, a także czy pozwoli na interpretację otrzymanych rezultatów. Wśród metod regresji wykorzystywanych do modelowania właściwości mechanicznych elastomerów można wyróżnić kilka głównych grup:

- **modele parametryczne** – proste w interpretacji, mają stałą liczbę parametrów niezależną od liczby obserwacji,
- **modele nieparametryczne** – umożliwiają lepszą reprezentację danych kosztem większej złożoności rosnącej wraz z liczbą obserwacji,

- **modele drzew decyzyjnych** – budują hierarchię prostych reguł dzielących przestrzeń cech. Dzielimy je na drzewa decyzyjne i ich zespoły, które łączą wyjścia z wielu drzew decyzyjnych, poprawiając dokładność i odporność na przeuczenie,
- **metody bayesowskie** – przewidują na podstawie rachunku prawdopodobieństwa, jednocześnie umożliwiając oszacowanie niepewności,
- **sieci neuronowe** – potrafią aproksymować dowolnie złożoną ciągłą funkcję, ale ze względu na ich złożoność są zwykle trudne w interpretacji.

W tab. 3.1 zamieszczono przykłady popularnych metod uczenia maszynowego.

Tabela 3.1. Podział metod regresji według klasy algorytmicznej

Model	**Kategoria**
LinearRegression	parametryczne
SVR	nieparametryczne
KNeighborsRegressor	nieparametryczne
DecisionTreeRegressor	drzewa decyzyjne
ExtraTreeRegressor	drzewa decyzyjne
RandomForestRegressor	zespoły drzew decyzyjnych
GradientBoostingRegressor	zespoły drzew decyzyjnych
AdaBoostRegressor	zespoły drzew decyzyjnych
BaggingRegressor	zespoły modeli
VotingRegressor	zespoły modeli
GaussianProcessRegressor	bayesowskie
MLPRegressor	sieci neuronowe

Celem niniejszego rozdziału jest opisanie wybranych metod regresji z algorytmicznego punktu widzenia. Opisy każdej z metod zawierają opis modelu matematycznego, główne hiperparametry, funkcję kosztu, sposób optymalizacji ich parametrów, a także przedstawiają wady i zalety każdej z nich.

Aby zapewnić możliwość porównania różnych metod, zaimplementowano ujednolicony przepływ danych, który używa klasy `Pipeline` z biblioteki `scikit-learn` [25] i kolejno wczytuje dane, przetwarza je, standaryzuje oraz dostraja hiperparametry, przeszukując siatkę z pięciokrotną walidacją krzyżową. Dane podzielono na treningowe i testowe z ustalonym ziarnem podziału w proporcji 80:20, co zapewnia powtarzalność wyników. Dla części modeli bazę cech rozszerzono o wyrażenia wielomianowe, wykorzystując klasę (`PolynomialFeatures`).

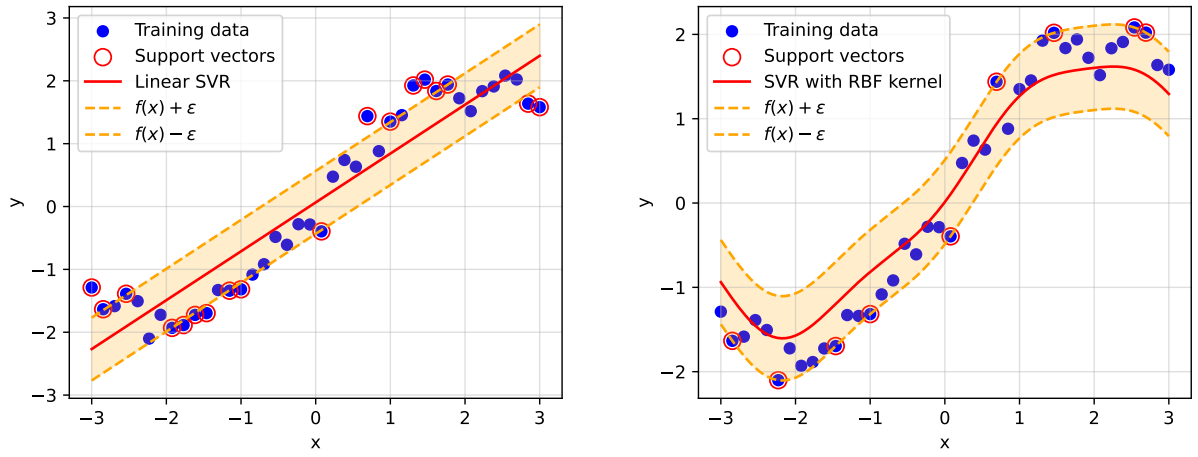
Po wybraniu najlepszego zestawu hiperparametrów określono metryki na zbiorze testowym, w których skład wchodzą błąd bezwzględny (MAE), pierwiastek z błędu śred-

niokwadratowego (RMSE) oraz współczynnik determinacji (R^2). Porównawcze stosowanie tej rodziny algorytmów, ocenianych miarami RMSE oraz R^2, jest w inżynierii lądowej powszechne, czego przykładem jest zestawienie maszyny wektorów nośnych, sieci neuronowych, i algorytmów drzewiastych w zadaniu szacowania temperatury warstw asfaltowych nawierzchni [26].

Omówione poniżej podejście ma pokazać, jak inżynier uczenia maszynowego nieposiadający wiedzy dziedzinowej z zakresu MOC mógłby podejść do problemu.

3.4.1. Maszyna wektorów nośnych

SVR (ang. Support Vector Regression, SVR) jest uogólnieniem maszyny wektorów nośnych [27] na zadanie regresji [28]. Zakłada, że w odpowiednio przekształconej (lub oryginalnej) przestrzeni cech istnieje hiperpłaszczyzna, która w optymalny sposób aproksymuje wartości zmiennej objaśnianej y. Optymalność ta realizowana jest poprzez minimalizację odległości pomiędzy hiperpłaszczyzną regresji a najbliższymi punktami zbioru treningowego przy założeniu jak najbardziej płaskiej hiperpłaszczyzny oraz minimalizację przekroczeń marginesu.



(a) SVR z kernelem liniowym

(b) SVR z kernelem gaussowskim

Rysunek 3.1. Regresor maszyny wektorów nośnych

W praktyce, w przypadku idealnego dopasowania, to właśnie obserwacje położone na granicy tuby ε lub ją przekraczające stanowią tzw. wektory nośne. Każda z takich obserwacji jest reprezentowana jako wektor w przestrzeni cech, stąd nazwa algorytmu – to one „niosą” informację o parametrach funkcji regresji, determinując jej położenie oraz nachylenie.

Sformułowanie problemu optymalizacyjnego SVR

Niech dane treningowe tworzy zbiór $\{(\underline{x}_i, \underline{y}_i)\}_{i=1}^n$, gdzie $\underline{x}_i \in \mathbb{R}^d$, a $\underline{y}_i \in \mathbb{R}^m$. W SVR chcemy odnaleźć funkcję:

$$\underline{f}(\underline{x}_i) \;=\; \underline{W}^\top \underline{x}_i \;+\; \underline{b}, \quad \underline{W} \in \mathbb{R}^{d\times m},\; \underline{b} \in \mathbb{R}^m, \tag{3.14}$$

która minimalizuje normę wag $\|\underline{W}\|$ przy jednoczesnym ograniczeniu odchyleń pomiędzy przewidywaniami $\underline{f}(\underline{x}_i)$ a wartościami $\underline{y}_i$. Sformułowanie SVR wyraża się jako problem optymalizacji:

$$J(\underline{W}, \underline{b}, \underline{\xi}) \;=\; \min_{\underline{W},\underline{b},\underline{\xi}} \left[\frac{1}{2}\,\|\underline{W}\|^2 \;+\; C\,\underline{1}^\top \underline{\xi}\right] \tag{3.15}$$

przy ograniczeniach dla $i = 1, \ldots, n$:

$$\left\| \underline{W}^\top \underline{x}_i + \underline{b} - \underline{y}_i \right\|_2 \;\leq\; \varepsilon + \xi_i, \qquad \xi_i \geq 0. \tag{3.16}$$

Parametr $C > 0$ stanowi kompromis pomiędzy płaskością funkcji (minimalizacją $\|\underline{W}\|^2$) a karami za odchylenia ξ_i wykraczające poza tubę tolerancji ε. Finalnie ogólna funkcja kosztu SVR jest połączeniem wielkości marginesu oraz kary jaką narzucamy za jej przekraczanie:

$$J(\underline{W}, \underline{b}) \;=\; \min_{\underline{W},\underline{b}} \left[\tfrac{1}{2}\,\|\underline{W}\|^2 + C\sum_{i=1}^{n} \max\left\{0,\; \left\| \underline{y}_i - (\underline{W}^\top \underline{x}_i + \underline{b}) \right\|_2 - \varepsilon\right\}\right]. \tag{3.17}$$

Kernel trick

W praktyce dane rzeczywiste często nie są liniowo separowalne ani dobrze aproksymowalne za pomocą liniowej hiperpłaszczyzny w przestrzeni oryginalnych cech. Aby rozszerzyć możliwości SVR na przypadki nieliniowe, stosuje się tzw. *kernel trick*. Idea polega na tym, aby dokonać zanurzenia:

$$\varphi : \mathbb{R}^d \;\longrightarrow\; \mathcal{H}, \qquad \underline{x} \;\mapsto\; \varphi(\underline{x}), \tag{3.18}$$

gdzie $\mathcal{H}$ jest przestrzenią z iloczynem skalarnym, w której dopiero zakładamy istnienie liniowej hiperpłaszczyzny regresyjnej. Jawne obliczanie $\varphi(\underline{x})$ bywa jednak kosztowne lub wręcz niewykonalne.

W przypadku regresji wielowymiarowej kluczowe jest uogólnienie tej idei: każdy wektor wag odpowiadający danej składowej wyjścia można przedstawić jako kombinację liniową wektorów zanurzonych danych treningowych. Oznacza to, że dla każdej kolumny $\underline{w}_k$ macierzy wag $\underline{W}$ zachodzi:

$$\underline{w}_k \;=\; \sum_{i=1}^{n} \alpha_{ik}\, \varphi(\underline{x}_i), \qquad \alpha_{ik} \in \mathbb{R}, \quad k = 1, \ldots, m, \tag{3.19}$$

gdzie n oznacza liczbę próbek treningowych, a m wymiar przestrzeni wyjściowej. Macierz $\underline{A} = [\alpha_{ik}] \in \mathbb{R}^{n \times m}$ przechowuje współczynniki kombinacji liniowych dla wszystkich wyjść jednocześnie.

Kluczowe jest to, że nie musimy znać samego zanurzenia: wystarczy umieć obliczać iloczyny skalarne odwzorowań, tzn.

$$k(\underline{x}_i, \underline{x}_j) \;=\; \langle \varphi(\underline{x}_i),\, \varphi(\underline{x}_j) \rangle. \tag{3.20}$$

Problem optymalizacji liniowego SVR (3.17) można przekształcić do postaci kernelowej:

$$J(\underline{A}, \underline{b}) \;=\; \min_{\underline{A} \in \mathbb{R}^{n \times m},\, \underline{b} \in \mathbb{R}^m} \left[\tfrac{1}{2} \operatorname{tr}(\underline{A}^\top \underline{K} \underline{A}) \;+\; C \sum_{i=1}^{n} \max \left\{ 0,\; \left\| \underline{y}_i - ((\underline{K}\,\underline{A})_{i,:} + \underline{b}) \right\|_2 - \varepsilon \right\} \right], \tag{3.21}$$

gdzie $\underline{K} = k(\underline{x}_i, \underline{x}_j) \in \mathbb{R}^{n \times n}$ jest macierzą Grama, a $(\underline{K}\underline{A})_{i,:}$ to i-ty wiersz iloczynu wewnętrznego macierzy $\underline{K}$ i $\underline{A}$.

Funkcja regresji przyjmuje wówczas postać

$$\underline{f}(\underline{x}) = \begin{bmatrix} \sum_{j=1}^{n} \alpha_{j1}\, k(\underline{x}_j, \underline{x}) + b_1 \\ \vdots \\ \sum_{j=1}^{n} \alpha_{jm}\, k(\underline{x}_j, \underline{x}) + b_m \end{bmatrix}. \tag{3.22}$$

Warto zauważyć, że w tej postaci wektory wag $\underline{W}$ nie występują jawnie — ich rolę przejmują kombinacje liniowe funkcji kernelowej. Parametry α_{jk} odpowiadają współczynnikom rozwinięcia wektorów wag w bazie zanurzonych próbek treningowych. Najczęściej stosowane kernele to:

- liniowe: $k(\underline{x}_i, \underline{x}_j) = \underline{x}_i^\top \underline{x}_j$,
- wielomianowe: $k(\underline{x}_i, \underline{x}_j) = (\gamma \underline{x}_i^\top \underline{x}_j + r)^d$,
- radialne: $k(\underline{x}_i, \underline{x}_j) = \exp(-\gamma \|\underline{x}_i - \underline{x}_j\|^2)$.

Wyniki

Przybliżenie związków konstytutywnych regresorem maszyny nośnej okazało się dość skuteczne. Wykresy naprężenie-odkształcenie dla zoptymalizowanych hiperparametrów z tab. 3.2 zaprezentowane na rys. 3.2 wyglądają poprawnie, a kernel umożliwia modelowanie nieliniowości bez konieczności wykonywania ręcznej ekstrakcji cech, choć przy zastosowaniu jego nieliniowych wariantów tracimy zdolność do interpretacji modelu, co jest pożądane w przypadku modelowania relacji konstytutywnych.

Wyjątkiem jest SVR z kernelem liniowym, którego postać modelu jest identyczna jak w przypadku liniowej regresji, niemniej jednak istnieją między nimi pewne różnice. Podczas gdy liniowy SVR minimalizuje (3.17), liniowa regresja minimalizuje sumę kwadratów reszt. SVR nie karze błędów znajdujących się w tubie tolerancji - w liniowej regresji wszystkie obserwacje będą brały udział w procesie optymalizacji, kiedy w SVR będą to tylko wektory nośne. Poza tubą tolerancji w przypadku SVR kara jest liniowa, a nie kwadratowa jak w liniowej regresji, więc rozwiązanie może być mniej wrażliwe na wartości odstające. Ponadto funkcja kosztu SVR posiada składnik regularyzujący, więc tak naprawdę jest bliższa regresji grzbietowej.

Zalety:

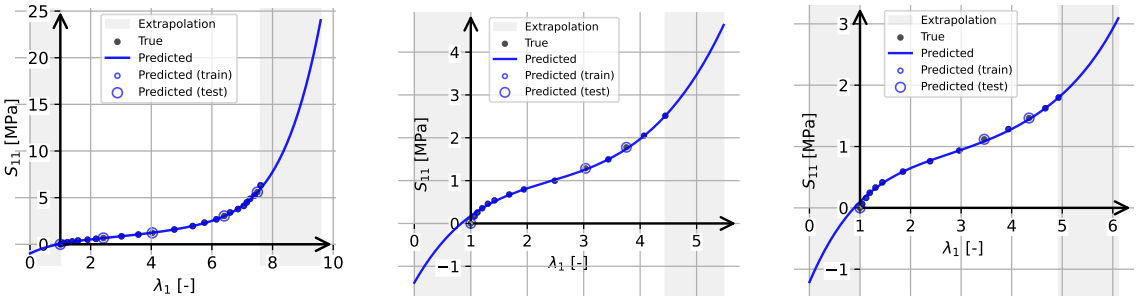
- dobre dopasowanie przy umiarkowanej liczbie obserwacji,
- kernel liniowy umożliwia pełną interpretowalność modelu,
- odporność na wartości odstające ze względu na karę poza tubą tolerancji,
- odporność na przeuczenie ze względu na składnik regularyzujący,
- niski koszt predykcji związany tylko z liczbą wektorów nośnych.

Ograniczenia:

- kernel nieliniowy umożliwia modelowanie skomplikowanych zależności kosztem interpretowalności,
- rosnący koszt uczenia przy dużych zbiorach ($\mathcal{O}(N^3)$ czasowo, $\mathcal{O}(N^2)$ pamięciowo).

Tabela 3.2. Strojenie hiperparametrów modelu `SVR` w procesie `GridSearchCV`. Pogrubieniem wyróżniono wartości optymalne.

Parametr	**Opis**	**Zakres wartości**
`C`	parametr regularyzacji	$\{0.001, 0.01, 0.1, \mathbf{1}, 10, 100\}$
`kernel`	funkcja kernelowa	{'linear', 'rbf', '**poly**', 'sigmoid'}
`degree`	stopień wielomianu przy kernelu 'poly'	$\{\mathbf{2}, 3, 4\}$
`gamma`	parametr kernela 'rbf','poly' i 'sigmoid'	{'scale', '**auto**'}
`epsilon`	tuba tolerancji	$\{\mathbf{0.0}, 0.1, 0.2\}$
`coef0`	niezależny wyraz dodawany w kernelach 'poly' i 'sigmoid'	$\{0.0, 0.1, 0.5, 1.0, \mathbf{2.0}\}$
`shrinking`	heurystyka przyspieszająca rozwiązanie	{True, **False**}



(a) Jednoosiowe rozciąganie **(b)** Dwuosiowe rozciąganie **(c)** Czyste ścinanie

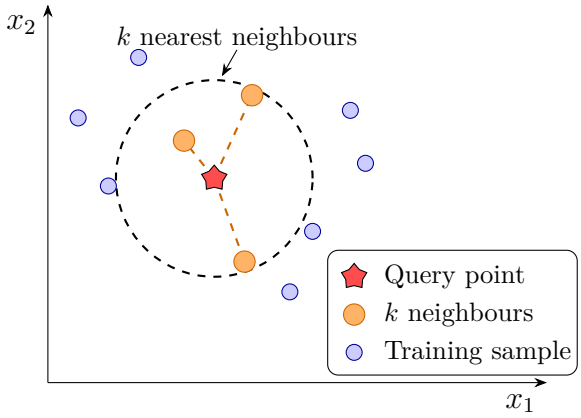
Rysunek 3.2. Wykresy S_{11} - wydłużenia λ_1 przedstawiające proste testy doświadczalne uzyskane z eksperymentu doświadczalnego przez Treloara (czarne punkty) i wyznaczonego przy użyciu maszyny wektorów nośnych z hiperparametrami z tabeli 3.2

Wykresy S_{11} względem wydłużenia λ_1: predykcje przy użyciu maszyny wektorów nośnych z hiperparametrami strojonymi wg tab. 3.2 na tle danych eksperymentalnych Treloara

3.4.2. Regresja *k*-najbliższych sąsiadów (*k*-NN)

Regresja k-NN jest klasycznym, nieparametrycznym algorytmem, w którym cała „wiedza" modelu przechowywana jest w zbiorze treningowym. Zamiast uczyć jawnej postaci funkcji regresji, estymuje się wartość wyjścia w punkcie zapytania $\underline{x}$ na podstawie wartości

obserwowanych w jego lokalnym sąsiedztwie w przestrzeni cech (rys. 3.3). Dzięki temu k-NN dobrze odwzorowuje lokalne nieliniowości i w naturalny sposób rozszerza się na przypadek wielowymiarowego wyjścia.



Rysunek 3.3. Schemat działania metody k najbliższych sąsiadów (k-NN) w dwuwymiarowej przestrzeni cech

Definicje i zapis macierzowy

Niech zbiorem treningowym będzie $\{(\underline{x}_i, \underline{y}_i)\}_{i=1}^n$, gdzie $\underline{x}_i \in \mathbb{R}^d$ oraz $\underline{y}_i \in \mathbb{R}^m$. Niech $d(\cdot,\cdot)$ oznacza metrykę w $\mathbb{R}^d$ oraz $\mathcal{N}_k(\underline{x}) \subset \{1,\ldots,n\}$ indeksy k najbliższych sąsiadów punktu $\underline{x}$ względem d. Regresor k-NN można zapisać jako ważoną średnią wektorów wyjściowych sąsiadów:

$$\widehat{\underline{f}}(\underline{x}) \;=\; \sum_{i\in\mathcal{N}_k(\underline{x})} w_i(\underline{x})\,\underline{y}_i, \qquad w_i(\underline{x}) \geq 0, \qquad \sum_{i\in\mathcal{N}_k(\underline{x})} w_i(\underline{x}) = 1. \tag{3.23}$$

Wybór wag definiuje konkretną odmianę estymatora:

- **jednorodne (klasyczne k-NN)**:

$$w_i(\underline{x}) = \frac{1}{k}, \tag{3.24}$$

- **odległościowe** ($p > 0$, $\delta > 0$ — regularyzacja):

$$w_i(\underline{x}) = \frac{\Big(d(\underline{x},\underline{x}_i) + \delta\Big)^{-p}}{\displaystyle\sum_{j\in\mathcal{N}_k(\underline{x})} \Big(d(\underline{x},\underline{x}_j) + \delta\Big)^{-p}}, \tag{3.25}$$

- **jądrowe / Nadaraya–Watson**:

$$w_i(\underline{x}) = \frac{K\left(\dfrac{d(\underline{x}, \underline{x}_i)}{h(\underline{x})}\right)}{\displaystyle\sum_{j \in \mathcal{N}_k(\underline{x})} K\left(\dfrac{d(\underline{x}, \underline{x}_j)}{h(\underline{x})}\right)}, \tag{3.26}$$

  gdzie K jest jądrem (np. gaussowskim), a $h(\underline{x})$ pełni rolę lokalnej szerokości pasma. Przyjęcie $h(\underline{x})$ równego odległości do k-tego sąsiada prowadzi do adaptacyjnej „szerokości okna".

W przypadku wyjścia wielowymiarowego $m > 1$ formuła (3.23) działa bez zmian: $\underline{y}_i$ są wektorami, a predykcja to ich ważona kombinacja. W zapisie macierzowym niech $\underline{Y}_{\mathcal{N}} \in \mathbb{R}^{k \times m}$ zawiera wierszami wektory $\underline{y}_i^{\top}$ sąsiadów, a $\underline{w} \in \mathbb{R}^k$ — wagi. Wówczas:

$$\widehat{\underline{f}}(\underline{x}) \;=\; \underline{Y}_{\mathcal{N}}^{\top}\, \underline{w} \;\in\; \mathbb{R}^m. \tag{3.27}$$

Metryki i skalowanie cech

Najczęściej stosuje się rodzinę metryk Minkowskiego

$$d_p(\underline{x}, \underline{z}) \;=\; \left(\sum_{j=1}^{d} |x_j - z_j|^p\right)^{1/p}, \quad p \in [1, +\infty), \tag{3.28}$$

w szczególności p=2 (euklidesową) oraz p=1 (Manhattan). Dla cech o różnych skalach konieczne jest standaryzowanie: $\tilde{x}_j = (x_j - \mu_j)/\sigma_j$, aby żadna cecha nie dominowała metryki. W obecności skorelowanych cech sensowne bywa użycie metryki Mahalanobisa $d_M(\underline{x}, \underline{z}) = \sqrt{(\underline{x} - \underline{z})^{\top} \underline{\Sigma}^{-1} (\underline{x} - \underline{z})}$, gdzie $\underline{\Sigma}$ to macierz kowariancji (np. oszacowana lokalnie).

Wyniki

Algorytm k-najbliższych sąsiadów (kNN) okazał się mało użyteczny w przeprowadzonych eksperymentach. Chociaż algorytm naturalnie rozszerza się na wielowymiarowe wyjście, co mogłoby być zaletą w przewidywaniu wszystkich składowych tensora naprężenia, zaleta ta została przytłumiona przez ograniczenie estymacji lokalnej - predykcje dały poszarpane wykresy naprężenie–wydłużenie i były zawodne w obszarach ekstrapolacji (por. rys. 3.4)

Zalety:

- łatwy w implementacji,
- łatwy w interpretacji,
- odporność na obserwacje odstające przy odpowiednio dużym k,
- naturalna obsługa wyjścia wielowymiarowego.

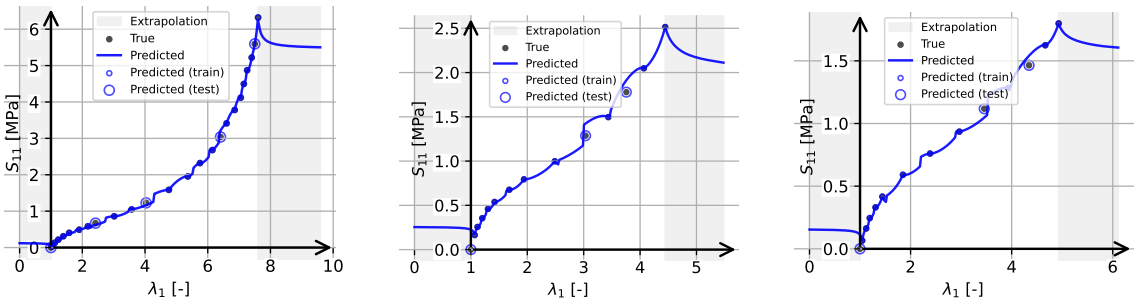
Ograniczenia:

- implementacja nie wymaga sformułowania ogólnego kształtu funkcji regresji przez co model nie ujawnia struktury zależności między zmiennymi,
- lokalna interpretowalność - algorytm wyjaśnia swoje przewidywania na podstawie k-najbliższych sąsiadów,
- wrażliwy na skalowanie zmiennych,
- wrażliwy na wybór metryki odległości,
- wysoki czas predykcji rosnący z rozmiarem zbioru,
- wymaga przechowywania całego zbioru treningowego na etapie predykcji,
- pogorszenie działania w wielu wymiarach (klątwa wymiarowości [29], [30]),
- niemożliwa ekstrapolacja.

Należy zauważyć, że łatwość w implementacji i interpretacji ma charakter ambiwalentny – brak założeń co do funkcji regresji, choć formalnie jest zaletą, to w przypadku modelowania związku konstytutywnego pozbawia nas możliwości globalnej interpretowalności i możliwości ekstrapolacji. W tabeli 3.3 przedstawiono zakres wartości testowanych dla poszczególnych parametrów.

Tabela 3.3. Strojenie hiperparametrów modelu `KNeighborsRegressor` w procesie `GridSearchCV`. Pogrubieniem wyróżniono wartości optymalne.

Parametr	Opis	Zakres wartości
`n_neighbors`	liczba sąsiadów uwzględnianych przy predykcji	$\{1, 3, 5, \mathbf{7}, 9, 11, 15\}$
`weights`	sposób ważenia sąsiadów	{'uniform', '**distance**'}
`metric`	metryka odległości	{`'euclidean'`, **`'minkowski'`**, `'manhattan'`}
`p`	wykładnik w metryce Minkowskiego	$\{1, \mathbf{2}, 3\}$
`algorithm`	algorytm wyszukiwania sąsiadów	{`'auto'`, `'ball_tree'`, **`'kd_tree'`**, `'brute'`}
`leaf_size`	rozmiar liścia dla struktur drzewa	$\{10, 20, \mathbf{30}, 40, 50\}$

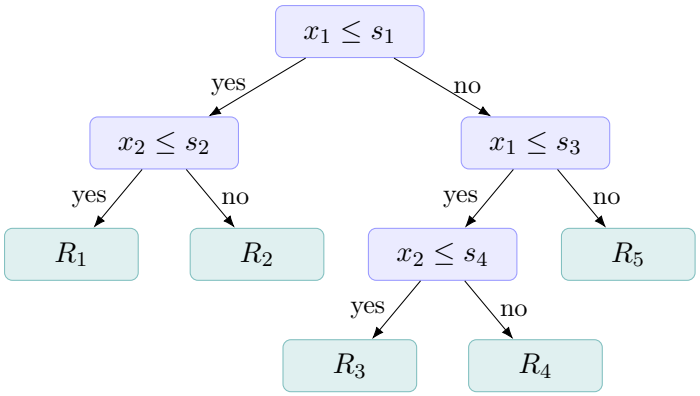


(a) Jednoosiowe rozciąganie **(b)** Dwuosiowe rozciąganie **(c)** Czyste ścinanie

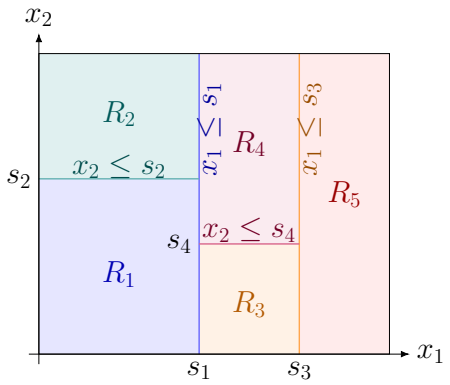
Rysunek 3.4. Wykresy S_{11} względem wydłużenia λ_1: predykcje algorytmu k-najbliższych sąsiadów z hiperparametrami strojonymi wg tab. 3.3 na tle danych eksperymentalnych Treloara

3.4.3. Regresyjne drzewo decyzyjne

Regresor drzewa decyzyjnego (ang. *Decision Tree Regressor*) konstruuje sekwencję reguł typu *jeśli–to*, które dzielą przestrzeń cech na rozłączne regiony, a następnie przypisują każdemu regionowi stałą wartość równą średniej z próbek w liściu(por. rys. 3.5). Model jest nieliniowy, łatwy do interpretacji i naturalnie uchwytuje interakcje między cechami poprzez hierarchiczne podziały.



(a) Przykładowa struktura drzewa z oznaczonymi węzłami i liśćmi



(b) Podział dwuwymiarowej przestrzeni cech na regiony liści

Rysunek 3.5. Regresyjne drzewo decyzyjne: przykład struktury i podziału przestrzeni cech

Sformułowanie modelu

Niech dane treningowe tworzą zbiór $\{(\underline{x}_i, \underline{y}_i)\}_{i=1}^{n}$, gdzie $\underline{x}_i \in \mathbb{R}^d$, a $\underline{y}_i \in \mathbb{R}^m$. Drzewo definiuje rozkład przestrzeni cech na rozłączne regiony $\{R_j\}_{j=1}^{J}$ (liście), zilustrowane na rys. 3.5a i 3.5b. Funkcja regresji ma postać skokową:

$$\underline{f}(\underline{x}) = \begin{cases} \underline{c}_1, & \underline{x} \in R_1, \\ \vdots & \\ \underline{c}_J, & \underline{x} \in R_J. \end{cases} \tag{3.29}$$

gdzie wektor przewidywany w liściu R_j wyznacza się jako średnią wektorów odpowiedzi należących do tego liścia:

$$\underline{c}_j = \frac{1}{n_j} \sum_{i:\, \underline{x}_i \in R_j} \underline{y}_i, \tag{3.30}$$

gdzie n_j oznacza liczbę próbek (obserwacji), które trafiły do liścia R_j.

Kryterium podziału

Niech $\underline{y}_i \in \mathbb{R}^m$, a średnią wektorów w węźle S oznacza się jako:

$$\bar{\underline{y}}_S = \frac{1}{n_S} \sum_{i \in S} \underline{y}_i, \tag{3.31}$$

oraz miarę niejednorodności w węźle jako średni kwadrat odchyleń wektorowych:

$$Q(S) \;=\; \frac{1}{n_S}\sum_{i\in S}\big(\underline{y}_i-\bar{\underline{y}}_S\big)^{\top}\big(\underline{y}_i-\bar{\underline{y}}_S\big) \;=\; \operatorname{tr}\Big(\widehat{\operatorname{Cov}}_S(\underline{y})\Big)\,, \tag{3.32}$$

gdzie $\widehat{\operatorname{Cov}}_S(\underline{y})$ oznacza empiryczną macierz kowariancji wektorów odpowiedzi w węźle S. Dla węzła S i podziału według cechy k oraz progu s definiujemy

$$S_L \;=\; \{\, i\in S:\; x_{ik}\le s\,\}, \qquad S_R \;=\; S\setminus S_L.$$

Optymalną parę (k^*,s^*) wybieramy minimalizując ważony błąd w węzłach potomnych (por. rys. 3.5b):

$$(k^*,s^*) \;=\; \arg\min_{k,s}\;\frac{n_{S_L}}{n_{S_t}}\,Q(S_L) \;+\; \frac{n_{S_R}}{n_{S_t}}\,Q(S_R). \tag{3.33}$$

Uwaga: Zastosowana miara niejednorodności węzła w (3.32) nie jest jedyna. W zależności od przyjętego kryterium może być zdefiniowana alternatywnie np. w oparciu o błąd absolutny, funkcję Hubera lub inne funkcje strat.

Zatrzymywanie wzrostu i przycinanie (regularyzacja)

Aby ograniczyć przeuczenie, regularyzację drzewa realizuje się dwojako: po pierwsze, poprzez warunki stopu (maksymalną głębokość, minimalną liczbę próbek w węźle lub liściu oraz minimalną poprawę kryterium podziału), które ograniczają złożoność już na etapie wzrostu; po drugie, poprzez przycinanie według minimum kosztu i złożoności (ang. *cost–complexity pruning*), które dla drzewa T i parametru $\alpha\ge 0$ polega na wyznaczeniu poddrzewa

$$T^* \;=\; \arg\min_{T\preceq T_{\max}}\Big[R(T)+\alpha\,|\tilde{T}|\Big] \tag{3.34}$$

gdzie $T_{\max}$ oznacza pełne (nieskrócone) drzewo, $R(T)$ jest łącznym błędem drzewa (np. sumą błędów w liściach), a $|\tilde{T}|$ liczbą jego liści. Drugi składnik pełni rolę kary za złożoność modelu, a hiperparametr α kontroluje kompromis pomiędzy dopasowaniem a złożonością. W praktyce wartość α dobiera się metodą walidacji krzyżowej, wybierając drzewo o minimalnym błędzie walidacyjnym.

Wyniki

W badaniach przeprowadzonych na rozpatrywanych testach materiałowych model regresyjny drzewa decyzyjnego (*Decision tree regressor*) również okazał się praktycznie mało przydatny. Choć drzewa dobrze odwzorowują nieliniowe zależności i lokalne interakcje cech, to przy niewielkich zbiorach pomiarowych wykazują tendencję do nadmiernego dopasowania (ang. *overfitting*) lub—po silnej regularyzacji—do niedoszacowania (ang. *underfitting*). Przewidywane charakterystyki naprężenie–wydłużenie są nieciągłe, schodkowe i stałe w obszarze ekstrapolacji (por. rys. 3.6), co dyskwalifikuje je do użycia podczas modelowania związków konstytutywnych.

Zalety:

- łatwy w implementacji,
- łatwy w interpretacji,
- naturalna obsługa wyjścia wielowymiarowego.

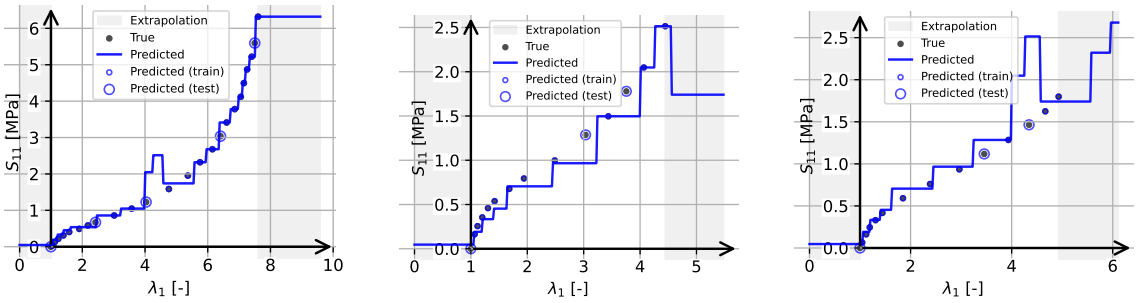
Ograniczenia:

- wrażliwość na wartości odstające,
- lokalna interpretowalność i nieciągłość funkcji regresji (schodkowość),
- silna zależność jakości od doboru parametrów regularyzacyjnych,
- pogorszenie działania w wielu wymiarach,
- niemożliwa ekstrapolacja.

Pomimo różnych mechanizmów przewidywania, drzewo decyzyjne i algorytm k-NN mają wiele wspólnego. Oba są proste w implementacji oraz interpretowalności i z ograniczeń z tego wynikających (nieciągłość, lokalna interpretowalność i niemożliwość ekstrapolacji) nie powinny być używane do modelowania relacji konstytutywnych. W tab. 3.4 zestawiono zakres testowanych hiperparametrów.

Tabela 3.4. Strojenie hiperparametrów modelu `DecisionTreeRegressor` w procesie `GridSearchCV`. Pogrubieniem wyróżniono wartości optymalne.

Parametr	Opis	Zakres wartości
`criterion`	funkcja kryterium podziału	{**`squared_error`**, `friedman_mse`}
`max_depth`	maksymalna głębokość drzewa	$\{3, \mathbf{5}, 8, 12\}$
`min_samples_split`	min. próbek do podziału węzła	$\{\mathbf{2}, 5\}$
`min_samples_leaf`	min. próbek w liściu	$\{\mathbf{1}, 2, 5\}$
`min_impurity_decrease`	min. spadek nieczystości wymagany do podziału	$\{\mathbf{0.0}, 10^{-7}, 10^{-5}\}$
`ccp_alpha`	siła przycinania o złożoności kosztu	$\{\mathbf{0.0}, 10^{-7}, 10^{-5}\}$



(a) Jednoosiowe rozciąganie **(b)** Dwuosiowe rozciąganie **(c)** Czyste ścinanie

Rysunek 3.6. Wykresy S_{11} względem wydłużenia λ_1: predykcje regresyjnego drzewa decyzyjnego z hiperparametrami strojonymi wg tab. 3.4 na tle danych eksperymentalnych Treloara

3.4.4. Las losowy

Regresor lasu losowego (ang. *random forest regressor*) to zespół (ang. *ensemble*) łączący n_t drzew decyzyjnych. Każde drzewo w lesie jest szkolone na losowej próbce danych (ang. *bootstrap sampling*) i podczas dokonywania podziału bierze pod uwagę tylko losowy podzbiór cech (ang. *feature randomization*). W przypadku zadań regresyjnych las uśrednia przewidywania wszystkich drzew. Siła modelu wynika z podejścia opartego na mądrości tłumów (ang. *wisdom of crowd*) – chociaż pojedyncze drzewa mogą popełniać

błędy, zbiorowy proces decyzyjny ma tendencję do uśredniania tych błędów i uzyskiwania bardziej wiarygodnych przewidywań.

Sformułowanie modelu

Niech dane treningowe tworzą zbiór $\{(\underline{x}_i, \underline{y}_i)\}_{i=1}^{n}$, gdzie $\underline{x}_i \in \mathbb{R}^d$, a $\underline{y}_i \in \mathbb{R}^m$. Las losowy jest średnią B niezależnie trenowanych drzew regresyjnych $\{f_b\}_{b=1}^{B}$:

$$\underline{f}(\underline{x}) \;=\; \frac{1}{B}\sum_{b=1}^{B} f_b(\underline{x}). \tag{3.35}$$

Każde drzewo f_b jest uczone na próbce $B^{(b)}$ losowanej ze zwracaniem z danych treningowych. Przy każdym kandydacie podziału rozważa się losowy podzbiór cech $\mathcal{F}_t^{(b)} \subset \{1, \ldots, d\}$ o rozmiarze m_{try}. W liściu $R_j^{(b)}$ predykcję:

$$\underline{c}_j^{(b)} \;=\; \frac{1}{n_j^{(b)}} \sum_{i:\, \underline{x}_i \in R_j^{(b)}} \underline{y}_i, \qquad f_b(\underline{x}) \;=\; \begin{cases} \underline{c}_1^{(b)}, & \underline{x} \in R_1^{(b)}, \\ \vdots & \\ \underline{c}_J^{(b)}, & \underline{x} \in R_J^{(b)} \end{cases} \tag{3.36}$$

i każdym węźle t podział (k, s):

$$(k^*, s^*) \;=\; \underset{k \in \mathcal{F}_t^{(b)},\, s}{\arg\min}\; \frac{n_{S_L}}{n_{S_t}}\, Q(S_L) + \frac{n_{S_R}}{n_{S_t}}\, Q(S_R) \tag{3.37}$$

otrzymujemy tak jak dla pojedynczego drzewa (por. (3.29) i (3.33)).

Estymacja OOB

Część obserwacji nie trafia do $B^{(b)}$ (*out-of-bag*, OOB). Każde drzewo jest testowane na własnych danych OBB (danych, które nie są wykorzystywane w jego szkoleniu). Uśredniając te indywidualne wyniki błędu na zbiorze OOB, las losowy zapewnia wbudowany sposób pomiaru wydajności bez konieczności stosowania osobnego zestawu testów (np. walidacji krzyżowej).

Regularyzacja i kompromisy

Las losowy reguluje się przez: liczbę drzew B (większe $B \Rightarrow$ mniejsza wariancja, większy koszt), ograniczenia złożoności drzew (głębokość, minimalne rozmiary liści) oraz m_{try} (im mniejszy, tym większa dekorrelacja drzew i zwykle lepsza generalizacja). W przeciwieństwie do pojedynczego drzewa, przycinanie typu cost–complexity zwykle nie jest konieczne (bagging + losowość cech pełnią rolę regularyzacji).

Wyniki

Na rozpatrywanych testach materiałowych `RandomForestRegressor` dawał przewidywania stabilniejsze od pojedynczego drzewa: uśrednienie wielu baz wygładza schodkowość i obniża wariancję, por.rys. 3.7. Funkcja regresji pozostaje jednak sumą funkcji skokowych, więc krzywe naprężenie–wydłużenie mają lokalne załamania, a ekstrapolacja poza zakres treningowy jest niemożliwa. Przy małych zbiorach i bardzo głębokich drzewach możliwe jest lekkie przeuczenie, choć mniej nasilone niż w pojedynczym drzewie.

Zalety:

- niska wariancja przewidywań względem pojedynczego drzewa dzięki baggingowi i losowości cech,
- odporność na wartości odstające,
- naturalna obsługa wyjścia wielowymiarowego,
- wbudowany mechanizm oceny wpływu cech.

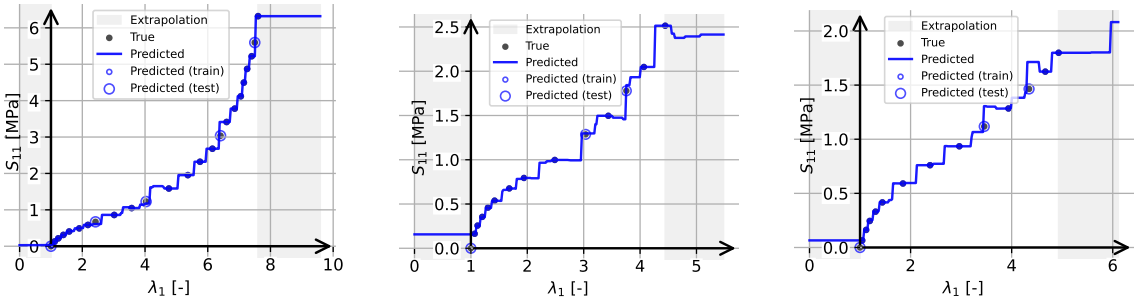
Ograniczenia:

- nieciągłość funkcji regresji,
- niemożliwa ekstrapolacja,
- pogorszenie działania w wielu wymiarach,
- mniejsza interpretowalność globalna względem pojedynczego drzewa,
- wysoki koszt obliczeń rosnący wraz z liczbą i głębokością drzew.

Zakres wartości testowanych parametrów i wybór optymalnych z nich zamieszczono w tab. 3.5.

Tabela 3.5. Strojenie hiperparametrów `RandomForestRegressor` w `GridSearchCV`. Pogrubieniem wyróżniono wartości optymalne.

Parametr	Opis	Zakres wartości
`n_estimators`	liczba drzew w lesie	$\{100, 300, \mathbf{600}, 1000\}$
`criterion`	kryterium podziału w drzewach	{**`squared_error`**, `friedman_mse`}
`max_depth`	maks. głębokość drzew	{**`None`**, 8, 12, 20}
`min_samples_split`	min. próbek do podziału	$\{2, \mathbf{5}, 10\}$
`min_samples_leaf`	min. próbek w liściu	$\{1, \mathbf{2}, 5\}$
`max_features`	liczba cech przy podziale	{`sqrt`, **`log2`**, 0.5, `None`}
`bootstrap`	losowanie ze zwracaniem	{**`True`**, `False`}
`oob_score`	estymacja błędu OOB	{`False`, **`True`**}
`min_impurity_decrease`	min. spadek nieczystości	$\{0,\ 10^{-7},\ \mathbf{10^{-5}}\}$



(a) Jednoosiowe rozciąganie **(b)** Dwuosiowe rozciąganie **(c)** Czyste ścinanie

Rysunek 3.7. Wykresy S_{11} względem wydłużenia λ_1: predykcje `RandomForestRegressor` ze strojeniem wg tab. 3.5 na tle danych eksperymentalnych Treloara

3.4.5. Liniowa regresja

Regresja liniowa jest jedną z najważniejszych i najszerzej stosowanych metod statystycznych oraz klasycznych metod uczenia maszynowego w analizie danych. Jej zastosowania są szerokie: od ekonomii i inżynierii po medycynę czy nauki społeczne, co dowodzi jej wszechstronności i przydatności w praktyce badawczej. Regresja liniowa stanowi podstawę wielu bardziej zaawansowanych algorytmów w uczeniu maszynowym, tym samym można ją wyróżnić jako jedno z istotnych narzędzi w sztucznej inteligencji.

Formalne nawiązanie do koncepcji regresji liniowej przypisuje się Francisowi Galtonowi, który w XIX wieku, badając związek wzrostu rodziców i ich dzieci, zwrócił uwagę na zjawisko „regresji do średniej" [31]. Jego badania w zakresie antropometrii stały się podstawą do opracowania metod regresji, a termin „regresja" został zapożyczony od tego zjawiska. Dalszego rozwoju tych pojęć matematycznych dokonał Karl Pearson, który podał formalną definicję współczynnika korelacji i skonkretyzował model regresji liniowej w ramach statystyki matematycznej [32].

Celem analizy regresji liniowej jest stworzenie modelu opisującego potencjalną liniową zależność między zmiennymi objaśnianymi a zmiennymi objaśniającymi, por. (3.1):

$$f_{\underline{W},\underline{b}}(\underline{x}) = \underline{b} + \underline{W}^{\top}\underline{x}, \qquad \underline{W} \in \mathbb{R}^{d\times m},\ \underline{b} \in \mathbb{R}^{m}, \tag{3.38}$$

gdzie $\underline{W}$ jest macierzą współczynników, a $\underline{b}$ wektorem wyrazów wolnych[21].

Parametry modelu określamy na podstawie zbioru treningowego, poszukując takich wartości współczynników macierzy $\underline{W}$ oraz $\underline{b}$, które sprawiają, że wyznaczone hiperpłaszczyzny (po jednej na każdą składową zmiennej objaśnianej) najlepiej aproksymują jej wartości. Najczęściej problem ten rozwiązujemy, wykorzystując sumę kwadratów błędów (ang. *residual sum of squares*) jako miarę dopasowania modelu, wyrażoną wzorem:

$$\mathrm{RSS}(\underline{W},\underline{b}) = \sum_{i=1}^{n} \left\| \underline{y}^{(i)} - f_{\underline{W},\underline{b}}(\underline{x}^{(i)}) \right\|_2^2. \tag{3.39}$$

W takim przypadku metoda najmniejszych kwadratów (MNK, ang. *least squares*) polega na wyznaczeniu:

$$(\hat{\underline{W}},\hat{\underline{b}}) = \arg\min_{\underline{W},\underline{b}} \mathrm{RSS}(\underline{W},\underline{b}). \tag{3.40}$$

Wyniki

Liniowa regresja okazała się najlepszą z klasycznych metod uczenia maszynowego do modelowania konstytutywnego. Jej głównym atutem jest prostota, możliwość stworzenia funkcji regresyjnej i możliwość budowania wyrażeń algebraicznych jako zmiennych wejściowych – jak przedstawiono w tab. 3.6 optymalny stopień wielomianu wyniósł 3. W tym kontekście w dalszej części pracy naprzemiennie używano terminów: baza wyrażeń algebraicznych, baza cech, baza funkcyjna, funkcje bazowe czy macierzy cech. Dzięki temu zabiegowi możemy modelować nieliniowe zależności przy zachowaniu liniowości względ-

dem bazy funkcyjnej. Model regresji liniowej został wzbogacony o wymagania mechaniki ośrodków ciągłych zestawione w rozdziale 5.8 oraz regularyzację i z tymi uwzględnieniami został rozwinięty w rozdziale 6. Wyniki predykcji przedstawiono na rys. 3.8. Uzyskana funkcja regresji jest ciągła i monotonicznie rosnąca oraz zdolna do ekstrapolacji. Widoczną niedociągnięciem jest brak założenia zerowania się składowych naprężeń w konfigurancji referencyjnej, które łatwo mogłoby być poprawione przez opcję `fit_intercept = False`, jednak na tym etapie celowo to pominięto, rezerwując wprowadzenie ograniczeń fizycznych do rozdziału 6.

Zalety:

- łatwa w implementacji i interpretacji,
- globalna interpretowalność - otrzymujemy zestaw parametrów odpowiadającym im funkcjom wejściowym,
- możliwość ekstrapolacji,
- naturalna obsługa wyjścia wielowymiarowego,
- niski koszt obliczeniowy.

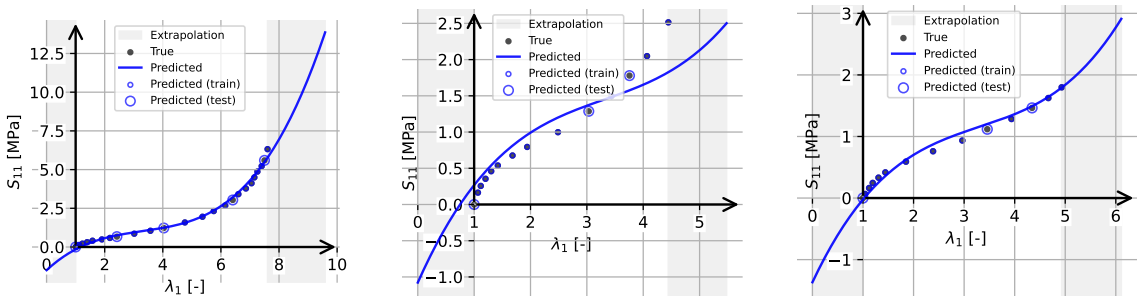
Ograniczenia:

- liniowość parametrów względem bazy funkcyjnej,
- jakość modelu zależna od wyboru bazy funkcyjnej,
- wrażliwość na wartości odstające,
- wrażliwość na współliniowość zmiennych.

Testowane hiperparametry i ich wartości optymalne zestawiono w tab. 3.6.

Tabela 3.6. Strojenie hiperparametrów regresji liniowej w `GridSearchCV`. Pogrubieniem wyróżniono wartości optymalne.

Parametr	**Opis**	**Zakres wartości**
`poly__degree`	stopień wielomianu cech	$\{1, 2, \mathbf{3}, 4\}$
`poly__include_bias`	człon stały w ekspansji wielomianowej	{**`True`**, `False`}
`fit_intercept`	wyraz wolny modelu	{**`True`**, `False`}
`positive`	wymuszenie nieujemności współczynników	{`True`, **`False`**}



(a) Jednoosiowe rozciąganie **(b)** Dwuosiowe rozciąganie **(c)** Czyste ścinanie

Rysunek 3.8. Wykresy S_{11} względem wydłużenia λ_1: predykcje `LinearRegression` ze strojeniem wg tab. 3.6 na tle danych eksperymentalnych Treloara

3.4.6. Wersjonowanie projektu

Wersjonowanie jest kluczowym aspektem zarządzania projektami uczenia maszynowego, umożliwiającym śledzenie zmian w modelach, zestawach danych i architekturach. Przyjęcie spójnej konwencji wersjonowania ułatwia utrzymanie kodu, zapewnia czytelność i przejrzystość, pracy a także pozwala zrozumieć zakres wprowadzanych aktualizacji. W tym rozdziale przedstawimy standardowy schemat wersjonowania oparty na konwencji major.minor.patch.build, począwszy od wersji 1.0.0.0.

Major - znaczące zmiany

Aktualizacje wersji podstawowej są zarezerwowane dla istotnych zmian, które mogą mieć wpływ na kompatybilność oraz podstawową funkcjonalność modelu. Przykłady zmian, które uzasadniają aktualizację wersji podstawowej obejmują:

- **modyfikacje zestawu danych** - przejście na całkowicie inny zestaw danych, który może wymagać dostosowania wstępnego przetwarzania danych i znacząco wpłynąć na wydajność modelu,
- **zmianę zadania** - zmiana głównego celu modelu, np. przejście od zadania regresji do zadania klasyfikacji.

Minor - ulepszenia funkcjonalne

Aktualizacje wersji drugorzędnej wprowadzają ulepszenia lub rozszerzenia, które dodają nowe możliwości bez zmiany podstawowej funkcjonalności modelu. Przykłady obejmują:

- **zmianę typu architektury** - wdrożenie innej architektury modelu w celu poprawy wydajności lub efektywności np. przejście z regresji liniowej na sieć neuronową.

Patch – konserwacja i optymalizacje

Przyrosty wersji poprawek obejmują drobne poprawki, optymalizacje i ulepszenia, które nie mają znaczącego wpływu na funkcjonalność modelu. Przykłady obejmują:

- **dostosowanie architektury** - wprowadzanie niewielkich modyfikacji do istniejącej architektury w celu zwiększenia wydajności lub stabilności np. zmiana z regresji Ridge na regresję Lasso w celu poprawy selekcji cech i możliwości interpretacji modelu,
- **poprawki błędów** - rozwiązywanie problemów w kodzie lub procesie uczenia w celu zapewnienia prawidłowego działania modelu.

Build - zmiany konfiguracji i hiperparametrów

Przyrosty wersji kompilacji odnoszą się do zmian w konfiguracji modelu, głównie obejmujących hiperparametry, które wpływają na jego wydajność bez zmiany jego struktury. Przykłady obejmują:

- **dostrajanie hiperparametrów** - dostosowywanie parametrów, takich jak np. wielkość kroku uczenia, liczba warstw w sieci neuronowej czy współczynniki regularyzacji, czy zmiana algorytmów optymalizacji w celu osiągnięcia lepszej zbieżności.

Opisany schemat wersjonowania zostanie praktycznie zastosowany w rozdziale 6, w którym rozwinięto dwa modele uczenia maszynowego – liniową regresję oraz sieci neuronowe – służące zarówno do wyznaczania parametrów klasycznych modeli funkcji jednostkowej energii sprężystości (JES), jak i do odkrywania nowych postaci tych funkcji. Oba modele były testowane iteracyjnie, uwzględniały wiele klasycznych modeli hipersprężystości w różnych konfiguracjach architektur. W związku z tym przyjęta konwencja *major.minor.patch.build* wersjonowania projektu zapewnia zdolność do śledzenia kolejnych etapów rozwoju modeli i identyfikować zakres wprowadzonej zmiany.

3.5. Podsumowanie

Przetestowano łącznie 13 algorytmów klasycznego uczenia maszynowego, z których żaden nie spełnił warunków modelowania konstytutywnego. Jak udowodniono w rozdziale 6 nie jest to możliwe bez wiedzy kierunkowej. Ponadto typowe dla uczenia maszynowego

metryki (RMSE, R^2) nie uwzględniają spełnienia więzów fizycznych i konieczności ekstrapolacji, co wskazuje na kierunek dalszych badań w tej płaszczyźnie. Interesującym podejściem byłoby pozostawienie jednej próby wytrzymałościowej w zbiorze testowym i wyznaczenie parametrów na pozostałych. W przypadku danych Treloara [33] implikuje to utratę $\frac{1}{3}$ danych. W praktyce eksperymentalnej rzadko dysponujemy większą ilością testów i chcemy z nich uzyskać jak najwięcej informacji – im więcej testów wytrzymałościowych uwzględnimy, tym aproksymacja funkcji jednostkowej energii sprężystości będzie pełniejsza. Z punktu widzenia uczenia maszynowego taka strata jest również bardzo duża (typowo na zbiór testowy rezerwuje się 10-20% danych). Wobec powyższego, w niniejszej pracy zaniechano takiego podziału i skupiono się na jak najlepszej aproksymacji przy wykorzystaniu wszystkich dostępnych prób wytrzymałościowych.

Zestawienie metryk na zbiorze testowym przedstawiono w tabeli 3.7, a pełny kod źródłowy i wyniki dostępne są w repozytorium `https://github.com/Mieczmik/elastomer-regression`. Kompletny przewodnik zastosowania rozpatrywanych metod do zadań klasyfikacji dostępny jest pod adresem `https://github.com/Mieczmik/titanic-kaggle-competition`.

Ze wszystkich sprawdzonych metod najlepszymi metodami do modelowania związków konstytutywnych hipersprężystości są regresja liniowa i regresor maszyny wektorów nośnych z jądrem liniowym (`LinearSVR`), które generują ciągłe funkcje regresji, są zdolne do ekstrapolacji i dają się interpretować. Co prawda najlepsze metryki osiągnął algorytm maszyny wektorów nośnych z jądrem wielomianowym, ale jego globalna interpretacja jest niemożliwa. Drugie miejsce zajęła sieć neuronowa `MLPRegressor`, której podstawom teoretycznym poświęcono rozdział 4, a jej wykorzystaniu w modelowaniu konstytutywnym rozdziały 6 i 8. Najsłabsze wyniki osiągnęły algorytmy drzewiaste (por. tab. 3.1), które nie poradziły sobie z interpolacją wartości ze zbioru treningowego na testowy.

Tabela 3.7. Porównanie metod uczenia maszynowego na zbiorze testowym

Model	**RMSE**	**MSE**	**MAE**	R^2
`SVR`	0.0674	0.0045	0.0470	0.9981
`MLPRegressor`	0.0897	0.0080	0.0641	0.9967
`KNeighborsRegressor`	0.1181	0.0139	0.1018	0.9943
`GradientBoostingRegressor`	0.1321	0.0174	0.1052	0.9929
`LinearRegression`	0.1360	0.0185	0.1122	0.9924
`VotingRegressor`	0.1440	0.0207	0.1072	0.9915
`LinearSVR`	0.1592	0.0254	0.1127	0.9896
`RandomForestRegressor`	0.1957	0.0383	0.1484	0.9843
`XGBRegressor`	0.2349	0.0552	0.1920	0.9775
`BaggingRegressor`	0.3605	0.1299	0.2708	0.9469
`ExtraTreeRegressor`	0.4128	0.1704	0.3245	0.9304
`DecisionTreeRegressor`	0.4566	0.2085	0.3332	0.9148
`AdaBoostRegressor`	0.4710	0.2218	0.3506	0.9093

4. Sieci neuronowe

4.1. Wstęp

Sieci neuronowe bywają postrzegane jako narzędzie skomplikowane i trudno dostępne, tymczasem ich konstrukcja opiera się na kilku prostych, wielokrotnie powtarzanych operacjach. Pojedynczy element sieci wykonuje jedynie ważoną sumę swoich wejść i wyznacza wartość prostej funkcji. Zdolność do odwzorowywania złożonych zależności bierze się dopiero z zestawienia wielu takich elementów oraz z doboru ich parametrów na podstawie danych. Celem niniejszego rozdziału jest przedstawienie tych podstaw w sposób przystępny również dla czytelnika niezaznajomionego z uczeniem maszynowym, tak aby działanie sieci wykorzystywanych w dalszej części pracy było dla niego zrozumiałe.

Wywód prowadzony jest od najprostszej jednostki do kompletnej procedury uczenia [1], [34], [35]. Rozpoczyna się od pojedynczego neuronu, czyli perceptronu (podrozdział 4.2), następnie omawia jego rozszerzenie do sieci wielowarstwowej (podrozdział 4.4), funkcje aktywacji oraz inicjalizację wag (podrozdział 4.3), a kończy na procedurze trenowania sieci wraz z technikami wspomagającymi jej zbieżność (podrozdział 4.5).

4.2. Perceptron

Pojedynczy neuron, zwany perceptronem, stanowi najprostszą jednostkę obliczeniową sztucznej sieci neuronowej. Idea sięga pracy Rosenblatta z 1958 r.[36], w której zaproponowano model inspirowany działaniem komórki nerwowej: na podstawie kilku sygnałów wejściowych jednostka generuje pojedynczą wartość wyjściową, regulując siłę odpowiedzi przez wagi przypisane poszczególnym wejściom oraz przez funkcję aktywacji rozstrzygającą, czy próg pobudzenia został przekroczony.

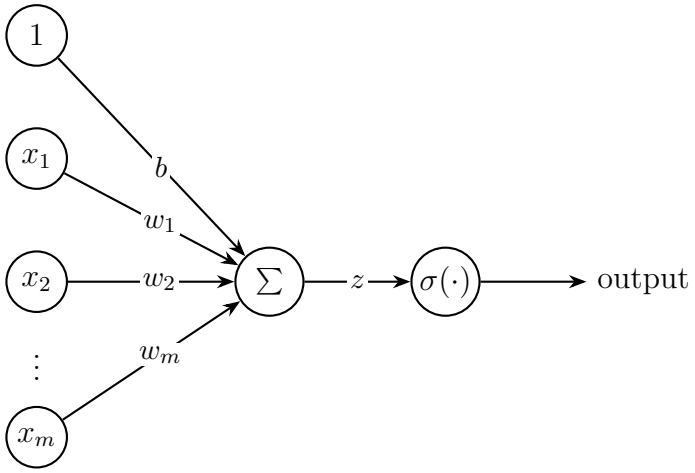
Niech $\underline{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ oznacza wektor wejściowy zawierający n skalarnych sygnałów. Pojedynczemu neuronowi przyporządkowuje się wektor wag $\underline{w} = (w_1, w_2, \ldots, w_n)^{\mathrm{T}}$ oraz skalarny wyraz wolny b, nazywany obciążeniem (ang. *bias*). Działanie neuronu rozkłada się na dwa etapy. Najpierw wyznacza się kombinację liniową wejść powiększoną o obciążenie:

$$z = \underline{w}^{\mathrm{T}}\underline{x} + b, \tag{4.1}$$

a następnie do tak otrzymanej wartości stosuje się nieliniową funkcję aktywacji $\sigma : \mathbb{R} \to \mathbb{R}$:

$$\hat{y} = \sigma(z) = \sigma\left(\underline{w}^{\mathrm{T}}\underline{x} + b\right). \tag{4.2}$$

Wartość $\hat{y}$ stanowi skalarne wyjście neuronu. Schemat działania pojedynczej jednostki przedstawiono na rysunku 4.1.



Rysunek 4.1. Schemat pojedynczego perceptronu

Konstrukcja ta uogólnia się w naturalny sposób na warstwę zawierającą wiele neuronów operujących na tym samym wektorze wejściowym. Jeżeli warstwa zawiera m jednostek, ich wagi zestawia się w macierz $\underline{W} \in \mathbb{R}^{n\times m}$, której kolejne kolumny odpowiadają kolejnym neuronom, a wyrazy wolne w wektor $\underline{b} \in \mathbb{R}^m$. Wyjście warstwy stanowi wówczas wektor $\underline{\hat{y}} \in \mathbb{R}^m$:

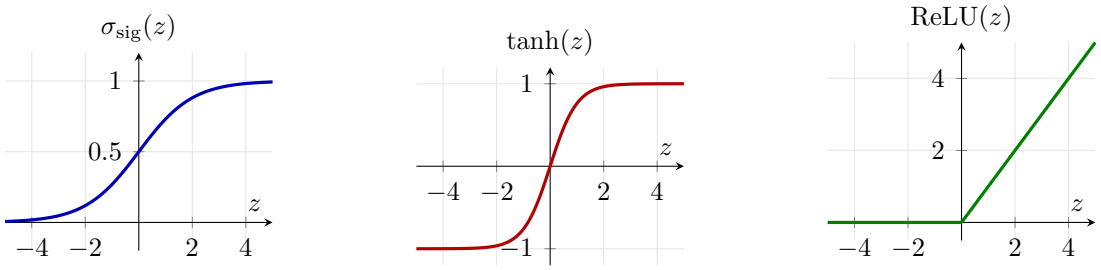
$$\underline{\hat{y}} = \sigma\left(\underline{W}^{\mathrm{T}}\underline{x} + \underline{b}\right), \tag{4.3}$$

przy czym funkcję aktywacji stosuje się składowo, niezależnie do każdej współrzędnej. Taka warstwa, w której każdy z neuronów połączony jest ze wszystkimi wejściami, nosi nazwę warstwy w pełni połączonej (ang. *fully connected* lub *dense*).

Kluczowym elementem konstrukcji perceptronu jest obecność funkcji aktywacji σ. Bez niej, tzn. przy $\sigma(z) = z$, złożenie kolejnych warstw redukowałoby się do pojedynczej transformacji liniowej, niezależnie od ich liczby. Każda funkcja całej sieci byłaby wtedy postaci $\underline{W}^{\mathrm{T}}\underline{x} + \underline{b}$, a model nie wykraczałby poza klasę regresji liniowej. Dopiero wprowadzenie nieliniowości w każdej warstwie pozwala sieci aproksymować szerokie klasy funkcji ciągłych. Szczegółowe omówienie funkcji aktywacji, ich własności oraz wpływu na zachowanie sieci stanowi przedmiot podrozdziału 4.3.

4.3. Funkcje aktywacji i inicjalizacja wag

Każda jednostka liniowa w sieci neuronowej zakończona jest funkcją aktywacji, której zadaniem jest wspomaganie procesu uczenia m.in. przez wygaszanie i wzmacnianie sygnałów.



(a) funkcja sigmoidalna **(b)** tangens hiperboliczny **(c)** funkcja ReLU

Rysunek 4.2. Standardowe funkcje aktywacji stosowane w sieciach neuronowych

Historycznie najwcześniej stosowano funkcję sigmoidalną [1]:

$$\sigma_{\text{sig}}(z) = \frac{1}{1+e^{-z}} \in (0;\, 1). \tag{4.4}$$

Pokrewną funkcją jest tangens hiperboliczny:

$$\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}} \in (-1;\, 1) \tag{4.5}$$

o wartościach wycentrowanych wokół zera. Funkcje (4.5) i (4.4) wypłaszczają się asymptotycznie dla dużych co do modułu argumentów, dla których ich pochodne dążą do zera. Przy złożeniu wielu warstw prowadzi to do problemu zanikających gradientów (ang. *vanishing gradients [1]*): sygnał propagowany wstecz przez kolejne warstwy zostaje wielokrotnie przemnożony przez wartości bliskie zera i wykładniczo zanika, uniemożliwiając aktualizację wag w warstwach najgłębszych. Tangens hiperboliczny jest najsilniej nieliniowy w okolicach zera (rysunek 4.2), co stanowi fundamentalną konsekwencję praktyczną dla sieci omawianych w rozdziałach 7 i 8: argumenty funkcji tanh muszą zawierać się w obszarze aktywnym (rzędu jedności), w przeciwnym razie neurony pracują w obszarze wypłaszczenia, gdzie również występuje zanikanie gradientów. Wymaganie to stanowi bezpośrednią motywację dla normalizacji zmiennych wejściowych, opisanej w podrozdziale 4.5.5.

Najpopularniejszą funkcją aktywacji w klasycznych zastosowaniach jest jednostka liniowa z progowaniem (ang. *rectified linear unit, ReLU [37]*):

$$\mathrm{ReLU}(z) = \max(0,\, z). \tag{4.6}$$

Jest ona prostsza i szybsza obliczeniowo niż funkcje nasycone, a sieci z ReLU zwykle zbiegają szybciej. Wadą jest natomiast problem martwych neuronów — jednostka, której wejście stale przyjmuje wartości ujemne, generuje na wyjściu zero, a gradient względem jej wag także znika. W odpowiedzi na to ograniczenie zaproponowano warianty zachowujące niezerowe nachylenie w obszarze ujemnym (Leaky ReLU, PReLU, ELU). W niniejszej pracy ReLU ani jej modyfikacje nie są wykorzystywane. W sieciach PINN konieczne jest obliczanie wyższych pochodnych wyjścia względem wejść, co wymaga gładkich funkcji aktywacji, stąd wybór funkcji tanh.

Sposób losowania wag startowych istotnie wpływa na zbieżność treningu. Glorot i Bengio [38] wykazali, że inicjalizacja z rozkładu o stałej wariancji prowadzi do zanikania lub eksplodowania gradientów w głębokich sieciach, a zaproponowana przez nich metoda dobiera wariancję wag tak, by wariancja sygnału oraz gradientu była zachowana pomiędzy warstwami. Aby zrozumieć, skąd biorą się konkretne wartości, warto prześledzić propagację wariancji przez pojedynczą warstwę liniową. Przy wejściach o jednakowej wariancji i wagach losowanych niezależnie ze średnią zero, wariancja każdej składowej wyjścia jest n_{in}-krotnością iloczynu wariancji wagi i wariancji wejścia. Zachowanie skali sygnału w przód wymaga zatem, by $n_{\mathrm{in}}\,\mathrm{Var}(W) \approx 1$, podczas gdy analogiczny warunek dla propagowanego wstecz gradientu daje $n_{\mathrm{out}}\,\mathrm{Var}(W) \approx 1$. Obu wymagań nie da się spełnić jednocześnie, więc Glorot i Bengio przyjmują kompromis uśredniający oba mianowniki:

$$\mathrm{Var}(W) = \frac{2}{n_{\mathrm{in}} + n_{\mathrm{out}}}. \tag{4.7}$$

W wariancie równomiernym wagi każdej warstwy o n_{in} wejściach i n_{out} wyjściach losuje się z rozkładu jednostajnego:

$$\underline{W} \sim \mathcal{U}\Big[-\sqrt{\tfrac{6}{n_{\mathrm{in}}+n_{\mathrm{out}}}},\; +\sqrt{\tfrac{6}{n_{\mathrm{in}}+n_{\mathrm{out}}}}\Big]. \tag{4.8}$$

Stała 6 pod pierwiastkiem wynika wprost z warunku (4.7). Dla rozkładu jednostajnego na przedziale $[-a, a]$ wariancja wynosi $a^2/3$, więc przyrównanie $a^2/3 = 2/(n_{\text{in}} + n_{\text{out}})$ daje $a = \sqrt{6/(n_{\text{in}} + n_{\text{out}})}$. Inicjalizacja ta jest dostosowana do funkcji aktywacji symetrycznych względem zera, takich jak tanh czy sigmoid i stanowi standardowy wybór w sieciach PINN wykorzystywanych w pracy — zarówno w rozdziale 7, jak i w rozdziale 8 wagi sieci inicjalizowane są zgodnie z wariantem `Glorot uniform`.

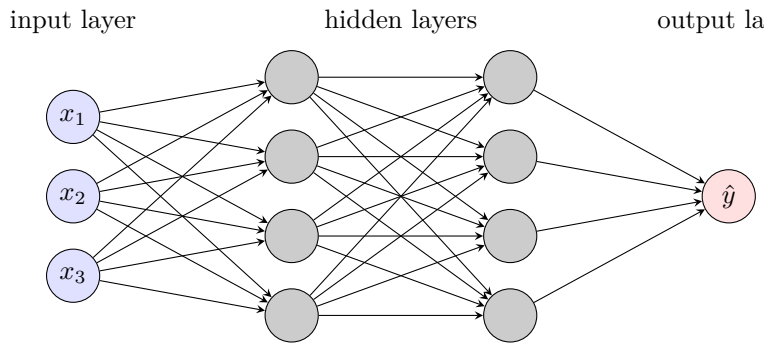
Założenie o symetrii aktywacji nie jest jednak spełnione dla funkcji ReLU, która wygasza całą ujemną połowę dziedziny. W efekcie połowa neuronów warstwy przekazuje na wyjściu zero, a wariancja sygnału po aktywacji spada o połowę względem przewidywań analizy liniowej. Dla takich sieci opracowano osobny schemat inicjalizacji He [39], w którym wagi losuje się z rozkładu normalnego, a wariancję dobiera się jedynie na podstawie liczby wejść:

$$\underline{W} \sim \mathcal{N}\left(0,\ \frac{2}{n_{\text{in}}}\right). \tag{4.9}$$

Stała 2 w liczniku kompensuje wygaszenie połowy sygnału przez ReLU: gdyby aktywacja zachowywała całą dziedzinę, wystarczyłaby wariancja $1/n_{\text{in}}$, lecz aby utrzymać skalę sygnału, należy ją podwoić. Pominięcie n_{out} odzwierciedla z kolei nacisk metody He na zachowanie skali sygnału propagowanego w przód.

4.4. Wielowarstwowy perceptron

Pojedyncza warstwa neuronów omówiona w rozdziale 4.2 oblicza ważoną sumę wejść powiększoną o obciążenie, a następnie podaje wynik po zastosowaniu funkcji aktywacji. Aby model potrafił odwzorowywać zależności nieliniowe, warstwy takie zestawia się jedna za drugą, tzn. wyjście jednej staje się wejściem kolejnej. Tak zbudowaną sieć nazywa się wielowarstwowym perceptronem (ang. *multilayer perceptron, MLP*) lub siecią jednokierunkową (ang. *feedforward neural network, FNN*). Gdy każdy neuron warstwy połączony jest ze wszystkimi neuronami warstwy poprzedniej, sieć jest w pełni połączona (ang. *fully connected feedforward neural network, FFNN*). Zaletą takiej konstrukcji jest to, że sieć samodzielnie buduje reprezentację danych odpowiednią dla rozwiązywanego problemu, bez ręcznego dobierania cech wymaganego w klasycznych metodach liniowych [1]. Przykładową sieć przedstawiono na rysunku 4.3.



Rysunek 4.3. Schemat w pełni połączonej sieci jednokierunkowej FFNN z dwiema warstwami ukrytymi

W sieci wyróżnia się trzy rodzaje warstw. Warstwa wejściowa (ang. *input layer*) nie wykonuje żadnych obliczeń. Wprowadza do sieci wektor danych opisujących pojedynczą obserwację, a jej szerokość równa jest liczbie zmiennych wejściowych. Warstwy ukryte (ang. *hidden layers*) leżą pomiędzy wejściem a wyjściem i kolejno przetwarzają sygnał, budując coraz bogatszą reprezentację danych. Ich liczba, nazywana głębokością sieci oraz liczba neuronów w każdej z nich, czyli szerokość, są hiperparametrami dobieranymi na etapie dostrajania architektury. Warstwa wyjściowa (ang. *output layer*) zwraca końcowy wynik, przy czym jej szerokość i funkcja aktywacji zależą od rodzaju zadania, co omówiono poniżej.

Niech sieć składa się z D warstw o szerokościach $n_0, n_1, \ldots, n_D$, gdzie n_0 jest wymiarem wejścia, a n_D — wyjścia. Każdej warstwie d przypisuje się wagi $\underline{W}^{(d)} \in \mathbb{R}^{n_{d-1} \times n_d}$ oraz obciążenia $\underline{b}^{(d)} \in \mathbb{R}^{n_d}$. Odpowiedź sieci wyznacza się warstwa po warstwie, co nazywa się propagacją w przód (ang. *forward pass*):

$$\underline{a}^{(0)} = \underline{x}, \tag{4.10a}$$

$$\underline{a}^{(d)} = \sigma\left(\underline{W}^{(d)\mathrm{T}} \underline{a}^{(d-1)} + \underline{b}^{(d)}\right), \qquad d = 1, \ldots, D-1, \tag{4.10b}$$

$$\underline{\hat{y}} = g\left(\underline{W}^{(D)\mathrm{T}} \underline{a}^{(D-1)} + \underline{b}^{(D)}\right). \tag{4.10c}$$

Warstwy o indeksach $d = 1, \ldots, D-1$ to warstwy ukryte (ang. *hidden layers*). Ich wartości nie są obserwowane bezpośrednio, lecz wykorzystywane jedynie wewnątrz sieci. W warstwach tych stosuje się jednakową funkcję aktywacji σ (w niniejszej pracy tangens hiperboliczny, podrozdział 4.3). Jej obecność jest niezbędna: bez nieliniowości złożenie kolejnych warstw sprowadzałoby się do zwykłego przekształcenia liniowego i sieć nie wykraczałaby poza model liniowy, co wykazano w podrozdziale 4.2.

Inaczej traktuje się funkcję wyjściową g w równaniu (4.10c), którą dobiera się do rodzaju zadania:

- **regresja** (przewidywanie zmiennej ciągłej) — g jest funkcją tożsamościową, dzięki czemu wyjście może przyjąć dowolną wartość rzeczywistą, a liczba neuronów odpowiada liczbie przewidywanych wielkości. Ten przypadek występuje w sieciach CANN i PINN rozważanych w niniejszej pracy,
- **klasyfikacja binarna** — g jest funkcją sigmoidalną (4.4), a pojedyncze wyjście interpretuje się jako prawdopodobieństwo przynależności do jednej z dwóch klas,
- **klasyfikacja wieloklasowa** — g jest funkcją *softmax*, dającą rozkład prawdopodobieństwa po K klasach (tyle też jest neuronów wyjściowych):

$$\mathrm{softmax}(\underline{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}. \tag{4.11}$$

Liczba parametrów podlegających uczeniu wynika wprost z przyjętej architektury. Zbierając wszystkie wagi i obciążenia w jeden wektor $\underline{\theta} = \{\underline{W}^{(d)}, \underline{b}^{(d)}\}_{d=1}^{D}$, a ich łączną liczbę $n_{\underline{\theta}}$ wyraża wzór:

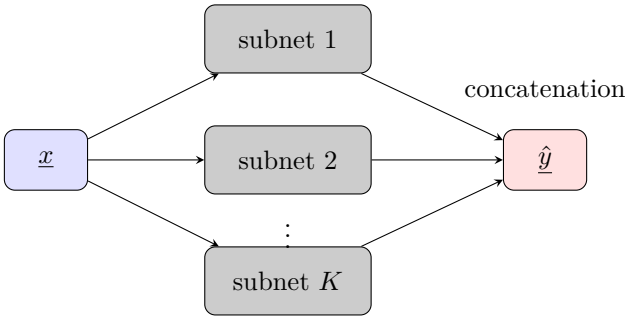
$$n_{\underline{\theta}} = \sum_{d=1}^{D} \left(n_{d-1}\, n_d + n_d\right). \tag{4.12}$$

Liczba parametrów rośnie więc wraz z szerokością warstw (iloczyn $n_{d-1}\, n_d$) oraz z ich liczbą, czyli głębokością sieci. Większa sieć ma większą zdolność do dopasowania, lecz podnosi koszt obliczeń i ryzyko przeuczenia (ang. *overfitting*), omawianego w podrozdziale 4.5.

Teoretyczne uzasadnienie zdolności aproksymacyjnych sieci daje twierdzenie o uniwersalnej aproksymacji Hornika, Stinchcombe'a i White'a [4], [40], zgodnie z którym już sieć z jedną warstwą ukrytą potrafi przybliżyć dowolną funkcję ciągłą f na zbiorze zwartym z dowolną dokładnością $\varepsilon > 0$, to jest $\sup_{\underline{x}} \|f(\underline{x}) - \underline{\hat{y}}(\underline{x})\| < \varepsilon$. Twierdzenie to dowodzi jednak jedynie istnienia takiej sieci i nie wskazuje, jak dobrać jej parametry ani ile neuronów jest potrzebnych. Ponieważ liczba neuronów w pojedynczej warstwie ukrytej potrafi rosnąć wykładniczo wraz ze złożonością problemu, w praktyce zamiast jednej bardzo szerokiej warstwy buduje się sieci głębokie, które przy porównywalnej liczbie parametrów osiągają lepsze wyniki i szybciej zbiegają podczas treningu [1], [35].

Architektura w pełni połączona nie jest jedynym możliwym wyborem. W niniejszej pracy wykorzystywany jest również wariant równoległej sieci jednokierunkowej (ang. *parallel feedforward neural network, PFNN*) [41], w którym każdą składową wyjścia oblicza

oddzielna podsieć przetwarzająca wspólne wejście, a ich wyniki się łączy (ang. *concatenation*)(rys. 4.4). Rozwiązanie to stosuje się w wariancie W3 sieci PINN z rozdziału 7, gdzie osobne gałęzie aproksymują poszczególne składowe pola przemieszczeń oraz w sieciach typu CANN z rozdziału 6, w których każdy niezmiennik tensora deformacji przetwarza własna gałąź.



Rysunek 4.4. Schemat równoległej sieci jednokierunkowej (PFNN). Wspólne wejście przetwarza K niezależnych podsieci, których wyjścia łączy się w jeden wektor wyjściowy.

4.5. Trenowanie sieci neuronowych

Trenowaniem sieci nazywa się procedurę doboru parametrów $\underline{\theta} = \{\underline{W}^{(d)}, \underline{b}^{(d)}\}_{d=1}^{D}$, minimalizującą funkcję kosztu $L(\underline{\theta})$ wyznaczoną na dostępnym zbiorze danych. Inicjalizację wag, ustalającą punkt startowy optymalizacji, omówiono w rozdziale 4.3 w kontekście odpowiadających jej funkcji aktywacji. W niniejszym rozdziale przedstawia się postać funkcji kosztu, algorytm wyznaczania jej gradientu, gradientowe metody optymalizacji wykorzystywane w pracy oraz techniki wspomagające zbieżność: normalizację wejść, regularyzację i wczesne zatrzymanie.

Iteracyjna procedura treningu sieci, której poszczególne składowe omawiają kolejne podsekcje niniejszego rozdziału, składa się z następujących kroków [35]:

1. Ustalenie hiperparametrów algorytmu, w tym kroku uczenia η i maksymalnej liczby iteracji n, zwanych epokami (ang. *epochs*),
2. Inicjalizacja parametrów $\underline{\theta}_0$ zgodnie ze schematem dobranym do funkcji aktywacji (rozdział 4.3), w PyTorch [42] realizowana automatycznie przy konstrukcji warstw klasy `torch.nn.Module`.
3. Powtarzanie dla $k = 0, 1, \ldots, n-1$:
   a) wyznaczenie wartości funkcji kosztu $L(\underline{\theta}_k)$ na podstawie propagacji w przód (4.10), w PyTorch realizowanej wywołaniem obiektu sieci na danych wejściowych,

b) wyznaczenie gradientu $\nabla_{\underline{\theta}} L(\underline{\theta}_k)$ algorytmem propagacji wstecznej, w PyTorch uruchamianej metodą `loss.backward()` korzystającą z modułu `autograd`. Przed każdym wyznaczeniem nowego gradientu konieczne jest wyzerowanie poprzednich wartości metodą `optimizer.zero_grad()`, ponieważ `autograd` akumuluje gradienty z kolejnych wywołań,

c) aktualizacja parametrów $\underline{\theta}_{k+1} = \underline{\theta}_k - \eta \, \nabla_{\underline{\theta}} L(\underline{\theta}_k)$, w PyTorch wykonywana przez metodę `step()` wybranego obiektu klasy `torch.optim.Optimizer`.

Powyższy zarys odpowiada najprostszej, klasycznej wersji algorytmu. Stosowane w praktyce warianty modyfikują przede wszystkim ostatni krok, wprowadzając pęd, adaptacyjny krok uczenia lub aproksymację drugich pochodnych, co omawia rozdział 4.5.3. Przykładową implementację opisanej pętli treningowej w środowisku PyTorch zamieszczono w załączniku 7.

4.5.1. Funkcja kosztu

Funkcja kosztu mierzy rozbieżność pomiędzy odpowiedziami sieci $\underline{\hat{y}}^{(i)}$ a wartościami docelowymi $\underline{y}^{(i)}$ wyznaczonymi na zbiorze treningowym o liczebności N:

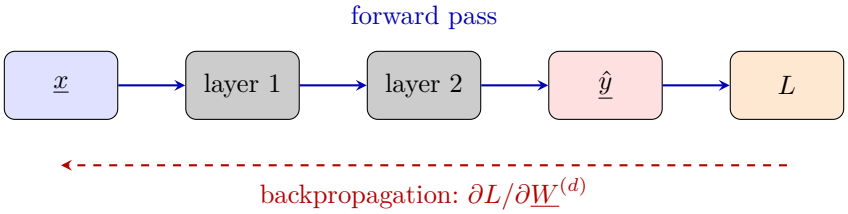
$$L(\underline{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \ell\left(\underline{\hat{y}}^{(i)}(\underline{\theta}),\, \underline{y}^{(i)}\right), \tag{4.13}$$

gdzie ℓ jest pojedynczym członem strat dobranym do rodzaju zadania. W zadaniach regresji standardowym wyborem jest średni błąd kwadratowy (MSE, por. (3.7)), który jest gładki i różniczkowalny w całej dziedzinie, co umożliwia stosowanie metod gradientowych, a dzięki zależności kwadratowej silnie karze duże odchylenia [43]. Średni błąd kwadratowy nie jest jednak jedynym możliwym wyborem, ponieważ postać członu ℓ można formułować dowolnie, w zależności od rozważanego zagadnienia i przyjętej koncepcji modelu. Dla porównania, w zadaniach klasyfikacji, niewystępujących wprawdzie w niniejszej pracy, stosuje się zwykle entropię krzyżową, lepiej dopasowaną do interpretacji wyjścia sieci jako rozkładu prawdopodobieństwa [43].

4.5.2. Algorytm propagacji wstecznej

Minimalizacja (4.13) metodami gradientowymi wymaga wyznaczenia pochodnych funkcji kosztu względem każdej wagi i obciążenia sieci. Liczba parametrów rośnie zgodnie z zależnością (4.12), a ich ręczne wyprowadzanie odrębnie dla każdej architektury

byłoby niepraktyczne. Standardowym rozwiązaniem jest algorytm propagacji wstecznej (ang. *backpropagation*), wykorzystujący regułę łańcuchową do rekurencyjnego wyznaczania pochodnych warstwa po warstwie, przechodząc przez sieć od wyjścia ku wejściu (rys. 4.5). Każda warstwa otrzymuje gradient kosztu względem swojego wyjścia obliczony w warstwie następnej i wykorzystuje go do wyznaczenia gradientu względem własnych parametrów oraz wejścia, które jest jednocześnie wyjściem warstwy poprzedniej.



Rysunek 4.5. Schemat propagacji w przód oraz propagacji wstecznej

Formalnie, oznaczając $\underline{z}^{(d)} = \underline{W}^{(d)\mathrm{T}}\underline{a}^{(d-1)} + \underline{b}^{(d)}$ argument funkcji aktywacji warstwy d oraz $\underline{\delta}^{(d)} = \partial L/\partial \underline{z}^{(d)}$ tzw. błąd warstwy, otrzymuje się dla warstw ukrytych zależność rekurencyjną:

$$\underline{\delta}^{(d)} = \left(\underline{W}^{(d+1)}\underline{\delta}^{(d+1)}\right) \odot \sigma'\left(\underline{z}^{(d)}\right), \tag{4.14}$$

gdzie σ' oznacza pochodną funkcji aktywacji, a $\odot$ iloczyn Hadamarda dwóch macierzy tych samych wymiarów $\underline{A} = [A_{ij}]$, $\underline{B} = [B_{ij}] \in \mathbb{R}^{m\times n}$ określony wzorem:

$$\underline{C} = \underline{A} \odot \underline{B} = [C_{ij}], \qquad \text{wskaźnikowo} \quad C_{ij} = A_{ij}\, B_{ij}, \qquad \begin{matrix} i = 1, \ldots, m, \\ j = 1, \ldots, n. \end{matrix} \tag{4.15}$$

Uwaga: Wyjątkowo nie obowiązuje tu konwencja sumacyjna — mimo powtórzonych wskaźników i, j nie sumujemy po nich, lecz otrzymujemy macierz iloczynów odpowiadających sobie składowych.

Gradienty względem parametrów warstwy d wyrażają się wówczas wprost:

$$\frac{\partial L}{\partial \underline{W}^{(d)}} = \underline{a}^{(d-1)}\underline{\delta}^{(d)\mathrm{T}}, \qquad \frac{\partial L}{\partial \underline{b}^{(d)}} = \underline{\delta}^{(d)}. \tag{4.16}$$

Wyznaczone pochodne (4.16) dla wszystkich warstw zestawia się w jeden wektor $\nabla_{\underline{\theta}} L$, którego składowe odpowiadają kolejnym składnikom $\underline{\theta} = \{\underline{W}^{(d)}, \underline{b}^{(d)}\}_{d=1}^{D}$ i stanowią wejście iteracyjnej procedury minimalizacji funkcji kosztu.

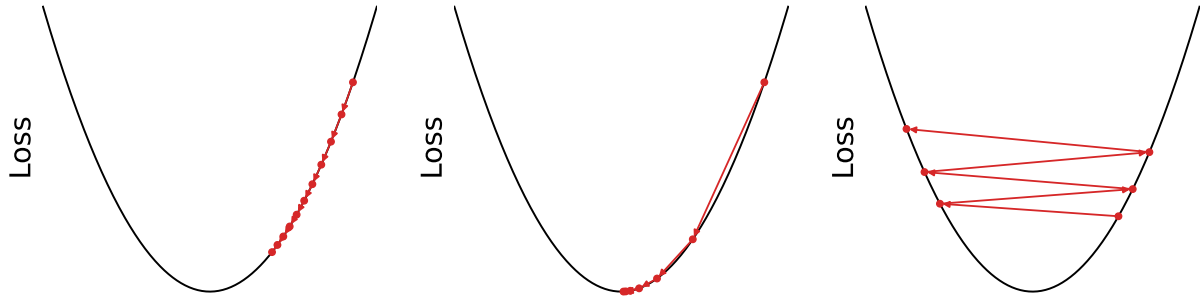
Łączny koszt obliczenia gradientu jest tego samego rzędu co koszt propagacji w przód, co czyni algorytm wykonalnym nawet dla sieci o milionach parametrów. W stosowanym w pracy środowisku PyTorch propagacja wsteczna realizowana jest przez moduł `autograd`, który automatycznie buduje graf obliczeń i wyznacza pochodne dowolnego wyjścia względem dowolnych zmiennych pośrednich. Mechanizm ten pozwala różniczkować wyjście sieci nie tylko względem jej parametrów, ale również względem wejść, na czym opiera się formułowanie reszt równań różniczkowych w sieciach PINN z rozdziałów 7 i 8 oraz wyznaczanie pochodnych funkcji jednostkowej energii sprężystości w sieciach CANN z rozdziału 6.

4.5.3. Gradientowe algorytmy optymalizacji

Algorytm propagacji wstecznej dostarcza gradientu funkcji kosztu, a zatem informacji o kierunku, w którym należy modyfikować parametry, aby ją zmniejszyć. Pozostaje wykorzystać tę informację w iteracyjnej procedurze poszukującej minimum. Punktem wyjścia większości stosowanych w praktyce metod jest stochastyczny spadek wzdłuż gradientu (ang. *stochastic gradient descent*, SGD). Aktualizacja parametrów w kroku k przyjmuje postać:

$$\underline{\theta}_{k+1} = \underline{\theta}_k - \eta \, \nabla_{\underline{\theta}} L(\underline{\theta}_k), \tag{4.17}$$

gdzie $\eta > 0$ jest krokiem uczenia (ang. *learning rate*). Dobór η ma istotny wpływ na zbieżność (rys. 4.6): zbyt małe wartości skutkują powolnym treningiem i podatnością na utknięcie w obszarach płaskich, zbyt duże — oscylacjami wokół minimum, a niekiedy rozbieżnością. Pojawiająca się w nazwie stochastyczność wynika ze sposobu, w jaki wyznacza się gradient $\nabla_{\underline{\theta}} L$ w każdej iteracji, co szczegółowo omawia podrozdział 4.5.4. W praktyce zamiast klasycznego SGD stosuje się jego usprawnione warianty łączące informację o historii gradientów z adaptacyjnym dostosowaniem kroku uczenia.



(a) zbyt mały **(b)** dobrany prawidłowo **(c)** zbyt duży

Rysunek 4.6. Wpływ wartości kroku uczenia η na zbieżność metody spadku wzdłuż gradientu dla przykładowej funkcji kosztu $L(\theta) = \theta^2$. Czerwone strzałki ilustrują kolejne aktualizacje parametru zgodnie ze wzorem (4.17).

W rzeczywistych zastosowaniach funkcja kosztu sieci neuronowej jest znacznie bardziej złożona niż przykład pokazany na rysunku 4.6. Określona w przestrzeni wielu parametrów, jest funkcją niewypukłą zawierającą liczne minima lokalne, punkty siodłowe oraz rozległe obszary płaskie, w których gradient niemal zanika. Klasyczna metoda SGD pozbawiona pamięci o poprzednich krokach łatwo utyka w takich obszarach lub oscyluje wzdłuż wąskich dolin, nie zbiegając do rozwiązania w rozsądnym czasie. Aby zaradzić tym trudnościom, do reguły aktualizacji wprowadza się dwa mechanizmy: pęd, akumulujący wcześniejsze gradienty i nadający aktualizacji bezwładność oraz adaptacyjny krok uczenia, dostosowywany osobno dla każdego parametru.

Najpowszechniej stosowanym z takich wariantów jest optymalizator Adam (ang. *Adaptive Moment Estimation*), zaproponowany przez Kingmę i Ba [44]. Wprowadza on oba mechanizmy poprzez wykładniczo wygaszane średnie pierwszego i drugiego momentu gradientu funkcji kosztu, na podstawie których wyznacza się aktualizację parametrów:

$$\begin{aligned} \underline{m}_k &= \beta_1 \underline{m}_{k-1} + (1-\beta_1)\,\nabla_{\underline{\theta}} L, \\ \underline{v}_k &= \beta_2 \underline{v}_{k-1} + (1-\beta_2)\,(\nabla_{\underline{\theta}} L) \odot (\nabla_{\underline{\theta}} L), \\ \underline{\theta}_{k+1} &= \underline{\theta}_k - \frac{\eta}{\sqrt{\underline{v}_k}} \odot \underline{m}_k, \end{aligned} \tag{4.18}$$

gdzie $\beta_1, \beta_2 \in [0, 1)$ są współczynnikami wygaszania (typowo $\beta_1 = 0.9$, $\beta_2 = 0.999$), $\sqrt{\cdot}$ wykonywane jest po każdej składowej. Pierwszy moment $\underline{m}_k$ pełni rolę pędu i uśrednia gradienty z poprzednich kroków. Drugi moment $\underline{v}_k$ uśrednia kwadraty gradientów, a dzielenie przez niego skaluje krok osobno dla każdego parametru.

Sposób aktualizacji opisany wzorem (4.18) wymaga jeszcze dwóch drobnych korekt. Na początku uczenia, dla małych k, średnie kroczące $\underline{m}_k$ i $\underline{v}_k$ pozostają silnie zaniżone względem wartości, które przyjęłyby przy dłuższym uśrednianiu. Powodem jest ich inicjalizacja zerami. Małe wartości $\underline{v}_k$ w mianowniku skutkują wtedy nieproporcjonalnie dużymi krokami, utrudniając zbieżność w pierwszych iteracjach. Aby temu zapobiec, wprowadza się następujące poprawki:

$$\begin{aligned} \underline{\hat{m}}_k &= \frac{\underline{m}_k}{1-\beta_1^k}, \\ \underline{\hat{v}}_k &= \frac{\underline{v}_k}{1-\beta_2^k}, \\ \underline{\theta}_{k+1} &= \underline{\theta}_k - \frac{\eta}{\sqrt{\underline{\hat{v}}_k} + \underline{\varepsilon}} \odot \underline{\hat{m}}_k, \end{aligned} \tag{4.19}$$

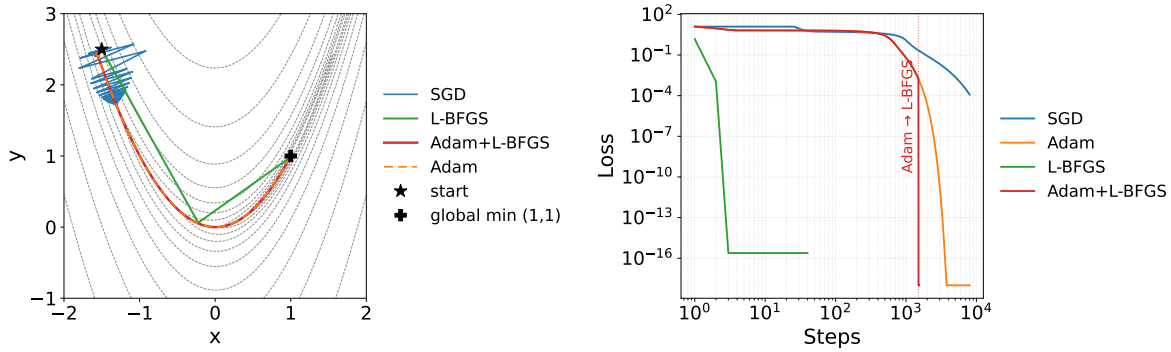
gdzie $\underline{\varepsilon} = \varepsilon\,\underline{1}$ jest niewielkim wektorem stabilizującym (typowo $\varepsilon = 10^{-8}$), zapobiegającym dzieleniu przez zero w mianowniku. Wzory (4.18) uzupełnione poprawkami (4.19) stanowią pełną definicję algorytmu Adam.

Odrębną klasę metod reprezentuje optymalizator L-BFGS (ang. (ang. *limited-memory Broyden-Fletcher-Goldfarb-Shanno*), zaproponowany przez Liu i Nocedala [45]. Jest to metoda drugiego rzędu i w odróżnieniu od opisanych wyżej algorytmów, które kierują się wyłącznie gradientem, wykorzystuje ona dodatkowo informację o krzywiźnie funkcji kosztu. Krzywiznę tę opisuje macierz drugich pochodnych funkcji kosztu, nazywana macierzą Hessego $\underline{H}_k$. Gdyby aktualizować parametry w oparciu o tę macierz, krok przyjmowałby postać:

$$\underline{\theta}_{k+1} = \underline{\theta}_k - \underline{H}_k^{-1}\,\nabla_{\underline{\theta}} L, \tag{4.20}$$

Jawne wyznaczenie i odwrócenie tej macierzy jest jednak niewykonalne dla sieci o dużej liczbie parametrów, gdyż jej rozmiar wynosi $n_{\underline{\theta}} \times n_{\underline{\theta}}$. Metoda BFGS zastępuje zatem dokładną odwrotność macierzy Hessego przybliżeniem budowanym iteracyjnie na podstawie kolejnych wartości gradientu oraz parametrów. Wariant L-BFGS (ang. *limited memory*) przechowuje przy tym jedynie ograniczoną liczbę ostatnich par przyrostów parametrów i gradientu zamiast całej macierzy, z których odtwarza działanie przybliżonej odwrotności na wektorze gradientu. W pobliżu minimum L-BFGS zbiega znacznie szybciej niż metody gradientowe pierwszego rzędu, jest jednak wrażliwy na punkt startowy i uruchomiony daleko od niego łatwo może się zatrzymać.

Z uzupełniających się własności obu algorytmów korzysta strategia dwuetapowa stosowana w sieciach PINN z rozdziałów 7 i 8. Jej przewagę nad pojedynczymi optymalizatorami pod względem prędkości zbieżności ilustruje porównanie na funkcji Rosenbrocka (rys. 4.7). W pierwszej fazie Adam, dzięki adaptacyjnej skali kroku, szybko sprowadza punkt początkowy z obszaru o dużej wartości funkcji kosztu do otoczenia minimum globalnego, nie wymagając przy tym precyzyjnego doboru hiperparametrów. W drugiej fazie L-BFGS, startując już blisko minimum, wykorzystuje informację o lokalnej krzywiźnie i osiąga niskie wartości kosztu w znacznie mniejszej liczbie kroków niż metody pierwszego rzędu.



(a) trajektorie optymalizacji **(b)** wartość funkcji kosztu w kolejnych krokach

Rysunek 4.7. Porównanie zbieżności SGD, Adama, L-BFGS oraz strategii dwuetapowej Adam+L-BFGS na funkcji Rosenbrocka

4.5.4. Uczenie wsadowe

Procedura aktualizacji parametrów (4.17) w swojej dosłownej postaci wymaga wyznaczenia gradientu funkcji kosztu na całym zbiorze treningowym w każdej iteracji. Dla typowych rozmiarów zbiorów danych jest to nieefektywne, a niekiedy także niewykonalne. Standardowym rozwiązaniem jest oszacowanie pełnego gradientu średnią gradientów liczonych na losowo wybranym podzbiorze obserwacji [46], nazywanym wsadem (ang. *batch*):

$$\nabla_{\underline{\theta}}\left(\frac{1}{N}\sum_{i=1}^{N}\ell^{(i)}\right) \approx \nabla_{\underline{\theta}}\left(\frac{1}{M}\sum_{i\in B}\ell^{(i)}\right), \tag{4.21}$$

gdzie N to liczba przykładów w zbiorze treningowym, B to bieżący wsad, a M to jego elementów. Przez $\ell^{(i)}$ skrótowo oznaczono $\ell\left(\underline{\hat{y}}^{(i)}(\underline{\theta}),\, \underline{y}^{(i)}\right)$ z równania (4.13).

Kolejne wsady przetwarza się sekwencyjnie: pierwsza iteracja wykorzystuje elementy $(\underline{x}^{(1)}, \underline{y}^{(1)}), \ldots, (\underline{x}^{(M)}, \underline{y}^{(M)})$, druga — $(\underline{x}^{(M+1)}, \underline{y}^{(M+1)}), \ldots, (\underline{x}^{(2M)}, \underline{y}^{(2M)})$ i tak dalej, aż do wyczerpania zbioru. Pełne przejście przez zbiór treningowy stanowi pojedynczą epokę.

Po jej zakończeniu kolejność elementów zbioru permutuje się losowo, aby kolejne epoki przetwarzały dane w innym porządku, i procedurę rozpoczyna się od nowa.

Aproksymacja (4.21) wprowadza do procesu uczenia element losowości. Wynikające z tego zaszumienie kroków, mimo pozornie niekorzystnego charakteru, w praktyce pełni rolę regularyzacyjną i sprzyja osiąganiu rozwiązań lepiej generalizujących na dane spoza zbioru treningowego.

4.5.5. Normalizacja danych wejściowych

Dobór efektywnego kroku uczenia jest tym trudniejszy, im bardziej składowe wektora wejściowego różnią się skalą. Składowe o dużych wartościach generują duże aktywacje, prowadząc do wysycenia funkcji tanh omawianej w rozdziale 4.3 i do zanikania gradientów. Składowe o małych wartościach przekazują zaś sygnał nieróżniący się od zera, co efektywnie wyłącza je z procesu uczenia. W typowym schemacie zmienne wejściowe sprowadza się do porównywalnej skali jedną z dwóch transformacji. Standaryzacja (ang. *z-score normalization*) zastępuje każdą zmienną jej wartością scentrowaną i podzieloną przez odchylenie standardowe:

$$\tilde{x}_j = \frac{x_j - \mu_j}{s_j}, \qquad \mu_j = \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)}, \qquad s_j = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(x_j^{(i)} - \mu_j\right)^2}. \tag{4.22}$$

Skalowanie min-max odwzorowuje dziedzinę każdej zmiennej w ustalony przedział, najczęściej $[-1, 1]$ lub $[0, 1]$:

$$\tilde{x}_j = \frac{x_j - \min_i x_j^{(i)}}{\max_i x_j^{(i)} - \min_i x_j^{(i)}}. \tag{4.23}$$

Po dowolnej z tych transformacji wszystkie składowe wejścia poruszają się w podobnym zakresie, dzięki czemu propagowane sygnały i gradienty zachowują porównywalną skalę pomiędzy współrzędnymi, a optymalizator może wykorzystywać wspólny krok uczenia. W sieciach PINN normalizacja nie jest jedynie zabiegiem usprawniającym, lecz warunkiem zbieżności, co wykazano w rozdziale 7.

4.5.6. Regularyzacja, przeuczenie i wczesne zatrzymanie

Sieci o dużej liczbie parametrów mają tendencję do nadmiernego dopasowywania się do zbioru treningowego (ang. *overfitting*), oddającego również szum pomiarowy zamiast właściwej zależności funkcyjnej. Klasycznym przeciwdziałaniem jest regularyzacja, czyli

rozszerzenie funkcji kosztu o człon karzący duże wartości wag. Regularyzacja L_2 (ridge, znana również jako ang. *weight decay*) [1], [47] dodaje sumę kwadratów wag:

$$L_2(\underline{\theta}) = L(\underline{\theta}) + \lambda \sum_{d=1}^{D} \left\| \underline{W}^{(d)} \right\|_2^2, \tag{4.24}$$

gdzie $\lambda \geq 0$ jest hiperparametrem określającym siłę regularyzacji. Człon (4.24) jest gładki i różniczkowalny, a jego pochodna zmniejsza każdą wagę proporcjonalnie do jej wartości, ograniczając rozrastanie się parametrów. Regularyzacja L_1 (lasso) [1], [47] zastępuje sumę kwadratów wag sumą ich wartości bezwzględnych:

$$L_1(\underline{\theta}) = L(\underline{\theta}) + \lambda \sum_{d=1}^{D} \left\| \underline{W}^{(d)} \right\|. \tag{4.25}$$

W odróżnieniu od (4.24), człon L_1 nie jest różniczkowalny w zerze, lecz dzięki temu wymusza dokładne zerowanie części wag, prowadząc do rozwiązań rzadkich. Obie regularyzacje wykorzystywane są w niniejszej pracy w rozdziale 6. Dobór odpowiedniej wartości λ stanowi tam jeden z etapów strojenia algorytmu.

Odrębną techniką, niewykorzystywaną wprawdzie w niniejszej pracy, lecz powszechnie stosowaną w głębokich sieciach, jest porzucanie neuronów (ang. *dropout*) [48]. W każdej iteracji treningu losowo wybraną część neuronów warstwy tymczasowo wyłącza się z obliczeń, ustawiając ich wyjścia na zero, przez co sieć nie może nadmiernie polegać na pojedynczych jednostkach i uczy się reprezentacji bardziej odpornych. Na etapie predykcji wszystkie neurony pozostają aktywne.

Diagnostyka jakości treningu opiera się na podziale dostępnych danych na trzy rozłączne podzbiory [1]. Na zbiorze treningowym wyznacza się parametry sieci. Zbiór walidacyjny nie uczestniczy w aktualizacji wag, lecz służy do monitorowania przebiegu treningu i doboru hiperparametrów. Zbiór testowy pozostaje natomiast całkowicie wyłączony z procedury uczenia i wykorzystuje się go wyłącznie do końcowej, jednorazowej oceny zdolności uogólniania gotowego modelu.

Przeuczenie objawia się rosnącą rozbieżnością wartości funkcji kosztu pomiędzy zbiorem treningowym a walidacyjnym: na zbiorze treningowym wartość ta maleje, podczas gdy na zbiorze walidacyjnym po pewnym czasie zaczyna rosnąć. Wczesne zatrzymanie (ang. *early stopping*) [1] polega na monitorowaniu wartości funkcji kosztu na zbiorze walidacyjnym i przerwaniu treningu w chwili, gdy ta przez ustaloną liczbę epok przestaje maleć. Ponieważ

moment zatrzymania zależy od zbioru walidacyjnego, ten ostatni wpływa na finalny model, co jest kolejnym powodem, dla którego niezależnej oceny dokonuje się dopiero na odłożonym zbiorze testowym. Sytuacja przeciwna, niedouczenie (ang. *underfitting*), w której wartość funkcji kosztu na obu zbiorach pozostaje wysoka, wskazuje na zbyt prostą architekturę sieci lub niewystarczająco długi trening.

5. Sformułowanie zagadnienia brzegowego w ramach mechaniki ośrodków ciągłych

W rozdziale tym przedstawiono podstawowe pojęcia i równania mechaniki ośrodków ciągłych niezbędne do sformułowania zagadnienia brzegowego nieliniowej teorii sprężystości. Szczegółowe omówienie poruszanych zagadnień, wraz z ich sformalizowanym ujęciem geometrycznym, zawierają podręczniki z mechaniki ośrodków ciągłych (MOC) [49]–[58] oraz monografie poświęcone nieliniowej teorii sprężystości i hipersprężystości [59]–[72] oraz obszerna literatura źródłowa tam cytowana.

W celu sformułowania zagadnienia brzegowego mechaniki ośrodków ciągłych przyjętym modelem matematycznym jest przestrzeń włóknista z wyróżnioną trójką elementów:

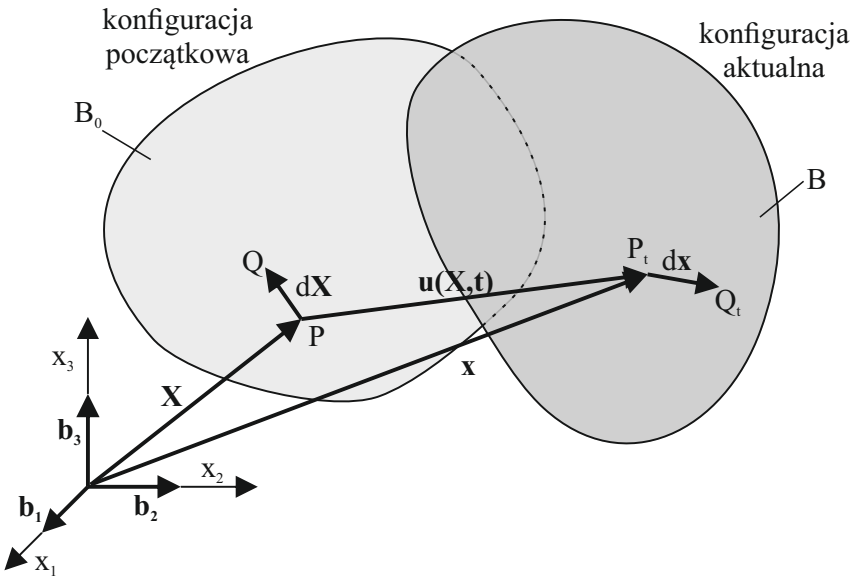
- przestrzenią wiązki, która jest trójwymiarową przestrzenią euklidesową punktową określającą położenie ciała,
- przestrzenią bazową, która jest jednowymiarową euklidesową punktową przestrzenią czasu,
- określoną surjekcją będącą odwzorowaniem rzutowania przestrzeni wiązki na przestrzeń bazową [73].

5.1. Deformacja

Aby rozróżnić konfigurację początkową B_0, odpowiadającą ciału nieodkształconemu, od konfiguracji aktualnej (ciała odkształcalnego) B, stosuje się materialny opis Lagrange'a bądź przestrzenny opis Eulera [73]. Za konfigurację odniesienia ciała nieodkształconego zwykle obiera się konfigurację początkową B_0, której przypisuje się chwilę $t = 0$. W ujęciu Lagrange'a dla chwil $t > 0$ położenie cząstki ciała wyznacza odwracalna wektorowa funkcja ruchu $\chi(\mathbf{X}, t)$ [56], [57], [61], [65]–[68], zależna od wektora położenia tej cząstki w konfiguracji odniesienia $\mathbf{X}$ oraz od czasu t. Z kolei opis Eulera, w którym argument $\mathbf{x}$ jest wektorem położenia cząstki w konfiguracji aktualnej, otrzymuje się przy użyciu funkcji odwrotnej $\chi^{-1}(\mathbf{x}, t)$. Prowadzi to do związku:

$$\mathbf{x} = \chi(\mathbf{X}, t) = \mathbf{X} + \mathbf{u}(\mathbf{X}, t), \tag{5.1}$$

przy czym $\mathbf{u}(\mathbf{X}, t)$ stanowi pole wektorowe przemieszczeń cząstki ciała, co ilustruje rys. 5.1. Ogólnie biorąc, każdej z konfiguracji towarzyszy pewien krzywoliniowy układ współ-



Rysunek 5.1. Początkowa oraz aktualna konfiguracja odkształcającego się ciała [71]

rzędnych. Szczególnym przypadkiem są układy kartezjańskie o odmiennych reperach, np. $\{O_0, \{\mathbf{E}_I\}\}$ w B_0 oraz $\{O, \{\mathbf{e}_i\}\}$ w B. Najprostsze podejście polega zaś na przyjęciu wspólnego dla obu konfiguracji układu kartezjańskiego, np. $\{O, \{\mathbf{e}_i\}\}$, por. rys. 5.1 [68]–[70], [74].

Dla odwzorowania $\chi(\mathbf{X}, t)$ wyznacza się pochodną Frécheta, zwyczajowo określaną mianem „gradientu deformacji":

$$\mathbf{F} \equiv \frac{\partial \chi(\mathbf{X}, t)}{\partial \mathbf{x}} = \mathbf{I} + \mathbf{H}, \mathbf{H} \equiv \frac{\partial \mathbf{u}(\mathbf{X}, t)}{\partial \mathbf{X}}. \tag{5.2}$$

Tensor $\mathbf{F}$ odwzorowuje sztywne obroty oraz wydłużenia infinitezymalnych nitek materialnych przy przejściu między obiema konfiguracjami [50], [64], [66], [67]:

$$d\mathbf{x} = \mathbf{F} d\mathbf{X}. \tag{5.3}$$

Z samej definicji tensora $\mathbf{F}$ płynie wniosek, że deformacje składają się multiplikatywnie:

$$\mathbf{F} = \mathbf{F}_1 \mathbf{F}_2 \mathbf{F}_3, \tag{5.4}$$

a operacja ta nie jest przemienna, czyli $\mathbf{F}_1 \mathbf{F}_2 \mathbf{F}_3 \neq \mathbf{F}_2 \mathbf{F}_1 \mathbf{F}_3$.

5.2. Tensory odkształceń

Kwadraty długości nitek materialnych odpowiednio w konfiguracji aktualnej oraz początkowej wynoszą:

$$dl^2 = d\mathbf{x}.d\mathbf{x}, \quad dL^2 = d\mathbf{X}.d\mathbf{X}, \tag{5.5}$$

co, korzystając z zależności 5.3, można rozwinąć do postaci:

$$\begin{aligned} dl^2 &= (\mathbf{F}d\mathbf{X}).(\mathbf{F}d\mathbf{X}) = (\mathbf{F}d\mathbf{X})^T(\mathbf{F}d\mathbf{X}) = d\mathbf{X}^T\mathbf{F}^T\mathbf{F}d\mathbf{X}, \\ dL^2 &= (\mathbf{F}^{-1}d\mathbf{x}).(\mathbf{F}^{-1}d\mathbf{x}) = (\mathbf{F}^{-1}d\mathbf{x})^T(\mathbf{F}^{-1}d\mathbf{x}) = d\mathbf{x}^T\mathbf{F}^{-T}\mathbf{F}^{-1}d\mathbf{x}. \end{aligned} \tag{5.6}$$

Pozwala to zinterpretować prawy $\mathbf{C}$ oraz lewy tensor Cauchy'ego–Greena $\mathbf{B}^{-1}$ [57], [65], [66], [68], [73] w postaci:

$$\mathbf{C} = \mathbf{F}^T\mathbf{F}, \quad \mathbf{c} = \mathbf{B}^{-1} = \mathbf{F}^{-T}\mathbf{F}^{-1} = \mathbf{F}\mathbf{F}^T \tag{5.7}$$

Objętość nieskończenie małego równoległościanu zbudowanego na trzech wektorach, zarówno przed deformacją, jak i po niej, otrzymuje się z ich iloczynu mieszanego:

$$dV = (d\mathbf{X}_1 \times d\mathbf{X}_2).d\mathbf{X}_3, \quad dv = (d\mathbf{x}_1 \times d\mathbf{x}_2).d\mathbf{x}_3. \tag{5.8}$$

Biorąc pod uwagę, że odwzorowanie nitek materialnych między obiema konfiguracjami realizuje tensor gradientu deformacji $\mathbf{F}$ [61], [65], [66], otrzymuje się:

$$dv = (\mathbf{F}d\mathbf{X}_1 \times \mathbf{F}d\mathbf{X}_2).\mathbf{F}d\mathbf{X}_3 = (\det\mathbf{F}) \cdot [(d\mathbf{X}_1 \times d\mathbf{X}_2).d\mathbf{X}_3] = (\det\mathbf{F})dV = JdV, \tag{5.9}$$

przy czym wykorzystano poniższe przekształcenia:

$$\begin{aligned} (\mathbf{F}d\mathbf{X}_1 \times \mathbf{F}d\mathbf{X}_2).\mathbf{F}d\mathbf{X}_3 &= \in_{ijk} [F_{ip}(dX_1)_p][F_{jq}(dX_2)_q][F_{kr}(dX_3)_r] \\ &= \in_{ijk} F_{ip}F_{jq}F_{kr}(dX_1)_p(dX_2)_q(dX_3)_r \\ &= \det\mathbf{F} \in_{pqr} (dX_1)_p(dX_2)_q(dX_3)_r \\ &= (\det\mathbf{F}) \cdot [(d\mathbf{X}_1 \times d\mathbf{X}_2).d\mathbf{X}_3] \end{aligned} \tag{5.10}$$

oraz przyjęto, że wyznacznik J, opisujący zmianę objętości, przyjmuje wartości dodatnie ($J > 0$). Wynika stąd, że dopuszczalne deformacje tworzą zbiór $\mathbf{F} \in Lin^+$, w którym Lin^+ oznacza przestrzeń dodatnio określonych tensorów drugiego rzędu ($\dim Lin^+ = 9$) [56],

[57], [61], [68]. Infinitezymalny element powierzchni z wersorem normalnym, rozpatrywany przed deformacją i po niej, wyraża się jako:

$$d\mathbf{S} = dS\,\mathbf{N}, \quad d\boldsymbol{s} = ds\,\mathbf{n}. \tag{5.11}$$

Rozważaną wcześniej objętość (5.8) zapisać można obecnie w formie:

$$dV = d\mathbf{S}.d\mathbf{X}, \quad dv = d\boldsymbol{s}.d\mathbf{x}. \tag{5.12}$$

Po skorzystaniu z (5.3) uzyskuje się:

$$d\boldsymbol{s}.d\mathbf{x} = J\,d\mathbf{S}.d\mathbf{X} = J\,d\mathbf{S}.(\mathbf{F}^{-1}d\mathbf{x}), \tag{5.13}$$

co po dalszych przekształceniach daje:

$$d\boldsymbol{s} = \mathbf{n}\,ds = J(\mathbf{F}^{-1})^T\mathbf{N}\,dS = (\mathrm{Cof}\,\mathbf{F})\mathbf{N}\,dS = (\mathrm{Cof}\,\mathbf{F})\,d\mathbf{S}. \tag{5.14}$$

Osobliwy tensor $\mathbf{F}$ poddaje się rozkładowi polarnemu [57], [65], [66]:

$$\mathbf{F} = \mathbf{RU} = \mathbf{VR}, \tag{5.15}$$

w którym $\mathbf{R}$ to ortogonalny tensor obrotu o jednostkowym wyznaczniku, natomiast $\mathbf{U}$ oraz $\mathbf{V}$ stanowią odpowiednio prawy i lewy dodatnio określony tensor wydłużeń, dany wzorami:

$$\mathbf{U} = \sqrt{\mathbf{C}} = \sqrt{\mathbf{F}^T\mathbf{F}}, \quad \mathbf{V} = \sqrt{\mathbf{B}} = \sqrt{\mathbf{F}\mathbf{F}^T}. \tag{5.16}$$

Tensory $\mathbf{C}, \mathbf{U}, \mathbf{B}, \mathbf{V}$ należą do Sym^+, czyli przestrzeni symetrycznych dodatnio określonych tensorów drugiego rzędu ($\dim Sym^+ = 6$). Dla tensorów $\mathbf{U}$ oraz $\mathbf{V}$ obowiązują twierdzenia spektralne:

$$\mathbf{U} = \sum_{k=1}^{K} \lambda_k \mathbf{P}_k, \quad \mathbf{V} = \sum_{k=1}^{K} \lambda_k \mathbf{p}_k, \tag{5.17}$$

przy czym λ_k oznaczają dodatnie wartości własne, nazywane wydłużeniami, zaś $\mathbf{P}_k$ i $\mathbf{p}_k$ są obróconymi wektorami własnymi odpowiednio w konfiguracji początkowej oraz aktualnej.

Analogicznie można napisać:

$$\mathbf{C} = \sum_{k=1}^{K} \lambda_k^2 \mathbf{P}_k, \quad \mathbf{B} = \sum_{k=1}^{K} \lambda_k^2 \mathbf{p}_k. \tag{5.18}$$

Gdy $\mathbf{C}$, $\mathbf{U}$, $\mathbf{B}$ i $\mathbf{V}$ redukują się do tensorów jednostkowych, gradient deformacji $\mathbf{F}$ reprezentuje wyłącznie sztywny obrót, co oznacza, że wymienione tensory charakteryzują czystą deformację. Posługiwanie się tensorami jednostkowymi do opisu stanu bez odkształceń jest jednak niewygodne, dlatego wprowadza się takie miary odkształcenia, dla których brak odkształceń odpowiada tensorowi zerowemu.

Przykładem takich miar jest rodzina tensorów odkształcenia Setha [57], [66], [68]:

$$\mathbf{E}^{(n)} = \frac{1}{n} \left(\mathbf{U}^n - \mathbf{I} \right). \tag{5.19}$$

W obrębie tej rodziny na uwagę zasługuje tensor odkształceń Lagrange'a, otrzymywany dla $n = 2$, tj. $\mathbf{E}^{(2)} = \mathbf{E}$, gdyż:

$$ds^2 - dS^2 = d\mathbf{X}^T \mathbf{F}^T \mathbf{F} d\mathbf{X} - d\mathbf{X}^T d\mathbf{X} = d\mathbf{X}^T (\mathbf{C} - \mathbf{I}) d\mathbf{X} = d\mathbf{X}^T (2\mathbf{E}) d\mathbf{X}. \tag{5.20}$$

Tensor Lagrange'a zapisuje się w postaci:

$$\mathbf{E} = \frac{1}{2} \left(\mathbf{F}^T \mathbf{F} - \mathbf{I} \right) = \frac{1}{2} \left[(\mathbf{H} + \mathbf{I})^T (\mathbf{H} + \mathbf{I}) - \mathbf{I} \right] = \frac{1}{2} \left(\mathbf{H} + \mathbf{H}^T + \mathbf{H}^T \mathbf{H} \right), \tag{5.21}$$

skąd, po przyjęciu wspólnej konfiguracji ($\mathbf{x} = \mathbf{X}$), wynika jego liniowe przybliżenie stosowane w TeMPO, czyli $\frac{1}{2}(\mathbf{H} + \mathbf{H}^T)$ [57], [65], [66], [68].

5.3. Gradient prędkości, tensor prędkości deformacji, tensor spinu

Na podstawie wektora położenia cząstki ciała można określić wektor prędkości i przyspieszenia:

$$\mathbf{v} = \frac{\partial \mathbf{x}(\mathbf{X}, t)}{\partial t} = \dot{\mathbf{x}}(\mathbf{X}, t), \quad \mathbf{a} = \frac{\partial \mathbf{v}(\mathbf{X}, t)}{\partial t} = \ddot{\mathbf{x}}(\mathbf{X}, t). \tag{5.22}$$

Gradient prędkości deformacji:

$$\mathbf{L} = \operatorname{grad} \mathbf{v}(\mathbf{x}, t) \tag{5.23}$$

przekształca gradient deformacji $\mathbf{F}$ w jego pochodną materialną $\dot{\mathbf{F}}$:

$$\dot{\mathbf{F}} = \mathbf{L}\mathbf{F}. \tag{5.24}$$

Rozkładając go na część symetryczną i antysymetryczną, otrzymujemy odpowiednio tensor prędkości deformacji $\mathbf{D}$ oraz tensor spinu $\mathbf{W}$ [57], [64], [67], [73]:

$$\mathbf{D} = \frac{1}{2}(\mathbf{L} + \mathbf{L}^T), \quad \mathbf{W} = \frac{1}{2}(\mathbf{L} - \mathbf{L}^T). \tag{5.25}$$

5.4. Tensory naprężenia i równania równowagi

Tensor naprężenia Cauchy'ego $\boldsymbol{\sigma}(\mathbf{x}, t)$, zdefiniowany w konfiguracji aktualnej, oraz tensor naprężenia Pioli-Kirchhoffa I rodzaju $\mathbf{S}(\mathbf{X}, t)$, odniesiony do konfiguracji początkowej, powiązane są zależnością [60], [64], [66], [73]:

$$\boldsymbol{\sigma}\mathbf{n}\, ds = \mathbf{S}\mathbf{N}\, dS. \tag{5.26}$$

Prowadzi to do zależności:

$$\mathbf{S} = J\boldsymbol{\sigma}\mathbf{F}^{-T} = J\boldsymbol{\sigma}\mathrm{Cof}\,\mathbf{F} = \boldsymbol{\tau}\mathrm{Cof}\,\mathbf{F}, \tag{5.27}$$

gdzie $\boldsymbol{\tau} = J\boldsymbol{\sigma}$ oznacza tensor naprężenia Kirchhoffa.

Zasady zachowania pędu oraz momentu pędu, zapisane w konfiguracji aktualnej, prowadzą do lokalnych równań [51], [64], [67], [68]:

$$\mathrm{div}\,\boldsymbol{\sigma} + \tilde{\mathbf{f}} = \rho\ddot{\mathbf{x}}, \quad \boldsymbol{\sigma} = \boldsymbol{\sigma}^T. \tag{5.28}$$

Ich odpowiednikiem w konfiguracji odniesienia są [50], [64]:

$$\mathrm{DIV}\,\mathbf{S} + \tilde{\mathbf{f}}_0 = \rho_0\ddot{\mathbf{x}}, \quad \mathbf{S}\mathbf{F}^T = \mathbf{F}\mathbf{S}^T, \tag{5.29}$$

gdzie symbole $\tilde{\mathbf{f}}$ oraz $\tilde{\mathbf{f}}_0$ reprezentują siły objętościowe w obu konfiguracjach. Drugie z równań (5.29) pokazuje, że tensor $\mathbf{S}$ pozostaje niesymetryczny, dlatego definiuje się

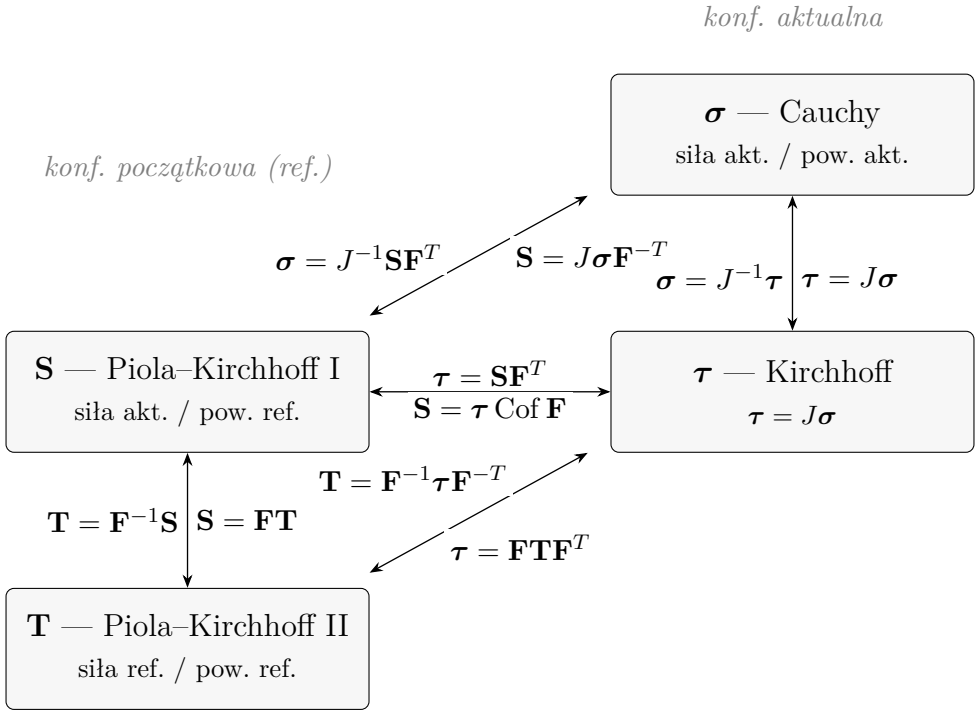
symetryczny tensor naprężenia Pioli-Kirchhoffa II rodzaju:

$$\mathbf{T} = \mathbf{F}^{-1}\mathbf{S}. \tag{5.30}$$

W przypadku quasi-statycznym ($\ddot{\mathbf{x}} = \mathbf{0}$) równanie (5.29) redukuje się do postaci:

$$\mathrm{DIV}\,\mathbf{S} + \tilde{\mathbf{f}}_0 = \mathbf{0}. \tag{5.31}$$

Powyższe równanie równowagi, wraz z odpowiednimi warunkami brzegowymi, stanowi zagadnienie brzegowe rozwiązywane w dalszej części pracy metodą PINN.



Rysunek 5.2. Relacje między tensorami naprężenia w teorii dużych deformacji

W niniejszej pracy relacje konstytutywne będą formułowane w kategoriach niezmienników tensorów Cauchy'ego–Greena, co prowadzi w sposób naturalny do tensora naprężenia Pioli-Kirchhoffa drugiego rodzaju $\mathbf{T}$, sprzężonego energetycznie z prawym tensorem Cauchy'ego–Greena $\mathbf{C}$. Na potrzeby identyfikacji parametrów oraz weryfikacji modelu wyniki przeliczane są na tensor naprężenia Pioli-Kirchhoffa pierwszego rodzaju $\mathbf{S}$, odpowiadający naprężeniu inżynierskiemu. Tensor ten pojawia się w danych doświadczalnych Treloara [33] oraz w naprężeniowych warunkach brzegowych metody PINN (ang. *Physics-Informed Neural Networks*) [2], [75].

5.5. Ogólna postać relacji konstytutywnych hipersprężystości

Łączne zastosowanie zasad zachowania pędu, momentu pędu, energii oraz masy — odpowiednio w konfiguracji aktualnej i odniesienia — prowadzi do związków [51], [64], [67], [68], [73]:

$$\rho\dot{U} = \boldsymbol{\sigma} \cdot \mathbf{D} = \mathrm{tr}(\boldsymbol{\sigma}\mathbf{D}), \quad \rho_0\dot{U} = \mathbf{S} \cdot \dot{\mathbf{F}} = \mathrm{tr}(\mathbf{S}\dot{\mathbf{F}}^T), \tag{5.32}$$

w których U jest energią wewnętrzną ciała, a $\mathbf{D}$ — tensorem prędkości deformacji. Dodatkowo definiuje się jednostkową energię sprężystości W:

$$\rho\dot{U} = \frac{\rho}{\rho_0}\dot{W} = \frac{1}{J}\dot{W} = \boldsymbol{\sigma} \cdot \mathbf{D} = \mathbf{S} \cdot \dot{\mathbf{F}} = \frac{1}{2J}\mathbf{T} \cdot \dot{\mathbf{C}} = \frac{1}{J}\mathbf{T} \cdot \dot{\mathbf{E}}, \tag{5.33}$$

z której, przy założeniu wystarczającej regularności i różniczkowalności funkcji JES, wyprowadza się relacje konstytutywne hipersprężystości:

$$\mathbf{S} = \nabla_{\mathbf{F}} W(\mathbf{F}) \equiv \left[\frac{\partial W(\mathbf{F})}{\partial \mathbf{F}}\right]^T, \ \mathbf{T} = \nabla_{\mathbf{E}} W(\mathbf{E}) \equiv \left.\frac{\partial W(\mathbf{E})}{\partial \mathbf{E}}\right|_{\mathbf{E}=\mathbf{E}^T} = 2 \left.\frac{\partial W(\mathbf{C})}{\partial \mathbf{C}}\right|_{\mathbf{C}=\mathbf{C}^T}. \tag{5.34}$$

Dla uproszczenia tę samą literę W wykorzystano do oznaczenia funkcji różniących się argumentami oraz dziedzinami [68]–[70], [74], [76].

5.6. Relacje konstytutywne jako funkcje niezmienników

W ogólnym ujęciu rolę argumentów funkcji jednostkowej energii sprężystości materiału izotropowego pełnić mogą trzy nieredukowalne, izotropowe niezmienniki tensorów wyznaczonych na podstawie $\mathbf{C}$ i $\mathbf{B}$. Mogą to być na przykład trzy dowolne niezmienniki dowolnej miary deformacji bądź odkształcenia (łącznie z ich wartościami własnymi). Wybór argumentów funkcji JES nie jest jednoznaczny, jednak gdy funkcja zależy od $\mathbf{C}$ i $\mathbf{B}$, ze względu na interpretację podstawowych niezmienników korzystnie jest przyjąć [68]–[70], [73], [74], [76]:

$$\begin{gathered} I_1 = \mathrm{tr}\,\mathbf{C} = \mathrm{tr}\,\mathbf{B}, \\ I_2 = \tfrac{1}{2}\left[(\mathrm{tr}\,\mathbf{C})^2 - \mathrm{tr}\,\mathbf{C}^2\right] = \mathrm{tr}\,\mathrm{Cof}\,\mathbf{C} = \tfrac{1}{2}\left[(\mathrm{tr}\,\mathbf{B})^2 - \mathrm{tr}\,\mathbf{B}^2\right] = \mathrm{tr}\,\mathrm{Cof}\,\mathbf{B}, \\ I_3 = \det\mathbf{C} = \det\mathbf{B} = J^2 > 0. \end{gathered} \tag{5.35}$$

Pierwszy niezmiennik I_1 jest związany z sumą kwadratów wydłużeń głównych ($I_1 = \lambda_1^2 + \lambda_2^2 + \lambda_3^2$) i opisuje średnią zmianę długości nitek materialnych. Drugi niezmiennik I_2 jest sumą kwadratów wydłużeń powierzchniowych ($I_2 = \lambda_1^2\lambda_2^2 + \lambda_2^2\lambda_3^2 + \lambda_3^2\lambda_1^2$) i charakteryzuje zmianę pól powierzchni elementów materialnych. Trzeci niezmiennik I_3 jest kwadratem jakobianu $J = \det \mathbf{F}$ ($I_3 = \lambda_1^2\lambda_2^2\lambda_3^2 = J^2$) i opisuje zmianę objętości elementu materialnego.

Przyjmijmy, że funkcja JES zależy od wprowadzonych powyżej niezmienników:

$$W(\mathbf{C}) = W(\mathbf{B}) = \breve{W}(I_1,\, I_2,\, I_3). \tag{5.36}$$

Pochodne tych niezmienników po tensorze $\mathbf{C}$ [51], [64], [67], [68]:

$$\frac{\partial I_1}{\partial \mathbf{C}} = \mathbf{I}, \quad \frac{\partial I_2}{\partial \mathbf{C}} = I_1\,\mathbf{I} - \mathbf{C}, \quad \frac{\partial I_3}{\partial \mathbf{C}} = I_2\,\mathbf{I} - I_1\,\mathbf{C} + \mathbf{C}^2. \tag{5.37}$$

Drugie z równań (5.34) prowadzi do związku konstytutywnego zapisanego w konfiguracji materialnej [55], [67], [77]:

$$\mathbf{T} = 2\,\frac{\partial \breve{W}}{\partial \mathbf{C}} = 2\Big(\gamma_1\,\mathbf{I} + \gamma_2\,\mathbf{C} + \gamma_3\,\mathbf{C}^2\Big)\,, \tag{5.38}$$

gdzie:

$$\gamma_1 = \alpha_1 + \alpha_2\,I_1 + \alpha_3\,I_2, \quad \gamma_2 = -(\alpha_2 + \alpha_3\,I_1), \quad \gamma_3 = \alpha_3, \tag{5.39}$$

natomiast współczynniki α_i stanowią pochodne funkcji JES:

$$\alpha_i = \frac{\partial \breve{W}}{\partial I_i}, \quad i = 1, 2, 3. \tag{5.40}$$

Wykorzystując związek $\boldsymbol{\sigma} = J^{-1}\mathbf{F}\mathbf{T}\mathbf{F}^T$ wraz z twierdzeniem Cayleya–Hamiltona, dochodzi się do analogicznego związku konstytutywnego w konfiguracji aktualnej [55], [67], [77]:

$$\boldsymbol{\sigma} = \frac{2}{J}\Big(\beta_1\,\mathbf{I} + \beta_2\,\mathbf{B} + \beta_3\,\mathbf{B}^2\Big)\,, \tag{5.41}$$

gdzie:

$$\beta_1 = \alpha_3\,I_3, \quad \beta_2 = \alpha_1 + \alpha_2\,I_1, \quad \beta_3 = -\alpha_2, \quad J = \sqrt{I_3}. \tag{5.42}$$

5.6.1. Rozkład objętościowo-izochoryczny

Gradient deformacji $\mathbf{F}$ poddaje się multiplikatywnemu rozkładowi na składową objętościową oraz izochoryczną [67], [78]:

$$\mathbf{F} = J^{1/3}\,\bar{\mathbf{F}}, \quad \det\bar{\mathbf{F}} = 1. \tag{5.43}$$

Funkcję JES zapisuje się wtedy następująco:

$$W = U\Big(\bar{I}_1,\ \bar{I}_2,\ J\Big), \tag{5.44}$$

przy czym niezmienniki deformacji izochorycznej [67], [78] przyjmują postać:

$$\bar{I}_1 = J^{-2/3}\,I_1, \quad \bar{I}_2 = J^{-4/3}\,I_2. \tag{5.45}$$

Definiuje się izochoryczny lewy tensor Cauchy'ego-Greena $\bar{\mathbf{B}} = J^{-2/3}\,\mathbf{B}$ wraz z odpowiadającymi mu dewiatorami:

$$\bar{\mathbf{B}}_D = \bar{\mathbf{B}} - \tfrac{1}{3}\,\bar{I}_1\,\mathbf{I}, \quad \bar{\mathbf{B}}_D^{-1} = \bar{\mathbf{B}}^{-1} - \tfrac{1}{3}\,\bar{I}_2\,\mathbf{I}. \tag{5.46}$$

Zmiana zmiennych w (5.41) prowadzi do związku konstytutywnego w konfiguracji aktualnej:

$$\boldsymbol{\sigma} = \frac{\partial U}{\partial J}\,\mathbf{I} + \frac{2}{J}\left(\frac{\partial U}{\partial \bar{I}_1}\,\bar{\mathbf{B}}_D - \frac{\partial U}{\partial \bar{I}_2}\,\bar{\mathbf{B}}_D^{-1}\right). \tag{5.47}$$

Przyjęcie stanu naturalnego ($\mathbf{E} = \mathbf{0}$) pociąga za sobą warunek $U(3,3,1) = 0$ [68].

Dodatkowo definiuje się izochoryczny prawy tensor Cauchy'ego–Greena $\bar{\mathbf{C}} = J^{-2/3}\,\mathbf{C}$ wraz z jego dewiatorami w konfiguracji materialnej:

$$\mathrm{DEV}\,\mathbf{I} = \mathbf{I} - \tfrac{1}{3}\,\bar{I}_1\,\bar{\mathbf{C}}^{-1}, \quad \mathrm{DEV}\,\bar{\mathbf{C}} = \bar{\mathbf{C}} - \tfrac{1}{3}\Big(\bar{I}_1^2 - 2\,\bar{I}_2\Big)\,\bar{\mathbf{C}}^{-1}. \tag{5.48}$$

Analogiczna zmiana zmiennych w (5.38) daje związek konstytutywny w konfiguracji materialnej:

$$\mathbf{T} = J^{1/3}\,\frac{\partial U}{\partial J}\,\bar{\mathbf{C}}^{-1} + J^{-2/3}\,\mathrm{DEV}\left(2\left(\frac{\partial U}{\partial \bar{I}_1} + \bar{I}_1\,\frac{\partial U}{\partial \bar{I}_2}\right)\mathbf{I} - 2\,\frac{\partial U}{\partial \bar{I}_2}\,\bar{\mathbf{C}}\right). \tag{5.49}$$

Szczegółowe wyprowadzenia relacji (5.47) i (5.49) zamieszczono w załączniku 1. Zestawienie relacji konstytutywnych w zależności od niezmienników I_1, I_2, I_3 oraz niezmienników izochorycznych $\bar{I}_1, \bar{I}_2, J$ przedstawiono w załączniku 2.

5.6.2. Klasyfikacja materiałów hipersprężystych

Dla funkcji JES o postaci (5.44) izotropowe materiały hipersprężyste dzieli się na [64], [67]:

a) nieściśliwe — funkcja JES nie zależy od niezmiennika opisującego zmiany objętości:

$$W = W_D\Big(\bar{I}_1,\ \bar{I}_2\Big)\,, \tag{5.50}$$

b) małościśliwe — funkcja JES stanowi sumę energii odkształcenia postaciowego oraz objętościowego:

$$W = W_D\Big(\bar{I}_1,\ \bar{I}_2\Big) + W_V(J), \tag{5.51}$$

c) ściśliwe — pozbawione powyższych ograniczeń, jak np. (5.44).

5.6.3. Materiały nieściśliwe

W materiałach nieściśliwych obowiązują więzy:

$$J = \det\mathbf{F} = 1 \quad \Longrightarrow \quad \det\mathbf{C} = 1 \ \wedge\ \det\mathbf{B} = 1, \tag{5.52}$$

pozostające w zgodzie z zasadą obiektywności. Funkcja JES przestaje wtedy pełnić rolę potencjału naprężeń, dlatego za potencjał sprężysty przyjmuje się funkcję Lagrange'a:

$$L\Big(\bar{I}_1,\ \bar{I}_2,\ J\Big) = W_D\Big(\bar{I}_1,\ \bar{I}_2\Big) - p\,(J-1), \tag{5.53}$$

w której p to mnożnik Lagrange'a utożsamiany ze średnim ciśnieniem hydrostatycznym. Jego wartość uzyskuje się dopiero po rozwiązaniu konkretnego zagadnienia brzegowego [68]. Z funkcji (5.53) otrzymuje się związek konstytutywny materiału nieściśliwego w konfiguracji aktualnej:

$$\boldsymbol{\sigma} = -p\,\mathbf{I} + \frac{2}{J}\left(\frac{\partial W_D}{\partial \bar{I}_1}\,\bar{\mathbf{B}} - \frac{\partial W_D}{\partial \bar{I}_2}\,\bar{\mathbf{B}}^{-1}\right). \tag{5.54}$$

5.6.4. Podstawowe testy doświadczalne materiałów nieściśliwych

Połączenie relacji konstytutywnej (5.54) z więzami nieściśliwości (5.52) prowadzi do związków łączących naprężenia nominalne z wydłużeniami próbki w trzech podstawowych testach doświadczalnych [33], [67], [73], [79]. Dla materiałów nieściśliwych czyste ścinanie odtwarza się eksperymentalnie poprzez próbę rozciągania, w której warunki brzegowe odpowiadają płaskiemu stanowi odkształcenia.

Występujące w tych testach naprężenie nominalne S_{11} odpowiada składowej pierwszego (niesymetrycznego) tensora naprężenia Pioli-Kirchhoffa $\mathbf{S} = J\,\boldsymbol{\sigma}\,\mathbf{F}^{-T}$, który figuruje w równaniach równowagi formułowanych w konfiguracji odniesienia.

a) Jednoosiowe rozciąganie ($\bar{\lambda}_2 = \bar{\lambda}_3 = \bar{\lambda}_1^{-1/2}$):

$$\bar{I}_1 = \bar{\lambda}_1^2 + 2\,\bar{\lambda}_1^{-1}, \quad \bar{I}_2 = \bar{\lambda}_1^{-2} + 2\,\bar{\lambda}_1, \tag{5.55}$$

$$S_{11} = \frac{\partial W\big(\bar{\lambda}_1\big)}{\partial \bar{\lambda}_1} = 2\Big(1 - \bar{\lambda}_1^{-3}\Big)\left(\bar{\lambda}_1\,\frac{\partial W_D}{\partial \bar{I}_1} + \frac{\partial W_D}{\partial \bar{I}_2}\right). \tag{5.56}$$

b) Dwuosiowe równomierne rozciąganie ($\bar{\lambda}_1 = \bar{\lambda}_2$, $\bar{\lambda}_3 = \bar{\lambda}_1^{-2}$):

$$\bar{I}_1 = 2\,\bar{\lambda}_1^2 + \bar{\lambda}_1^{-4}, \quad \bar{I}_2 = 2\,\bar{\lambda}_1^{-2} + \bar{\lambda}_1^4, \tag{5.57}$$

$$S_{11} = S_{22} = \tfrac{1}{2}\,\frac{\partial W\big(\bar{\lambda}_1\big)}{\partial \bar{\lambda}_1} = 2\Big(\bar{\lambda}_1 - \bar{\lambda}_1^{-5}\Big)\left(\frac{\partial W_D}{\partial \bar{I}_1} + \bar{\lambda}_1^2\,\frac{\partial W_D}{\partial \bar{I}_2}\right). \tag{5.58}$$

c) Czyste ścinanie — realizowane jako jednoosiowe rozciąganie w PSO ($\bar{\lambda}_2 = 1$, $\bar{\lambda}_3 = \bar{\lambda}_1^{-1}$):

$$\bar{I}_1 = \bar{I}_2 = \bar{\lambda}_1^2 + \bar{\lambda}_1^{-2} + 1, \tag{5.59}$$

$$S_{11} = \frac{\partial W\big(\bar{\lambda}_1\big)}{\partial \bar{\lambda}_1} = 2\Big(\bar{\lambda}_1 - \bar{\lambda}_1^{-3}\Big)\left(\frac{\partial W_D}{\partial \bar{I}_1} + \frac{\partial W_D}{\partial \bar{I}_2}\right). \tag{5.60}$$

Do równań (5.56), (5.58) i (5.60) wstawia się relację konstytutywną (5.54), uwzględniając jednocześnie odpowiednie niezmienniki izochoryczne (5.55), (5.57) oraz (5.59). [67], [79]

5.6.5. Aspekty implementacyjne w MES

W praktyce obliczeniowej komercyjne solvery MES (m.in. Abaqus [80], ANSYS [81], Marc [82]) stosują postać małościśliwą (5.51) lub nieściśliwą (5.50). Addytywny rozkład na

W_D i W_V w wariancie małościśliwym umożliwia osobne traktowanie części izochorycznej i objętościowej na poziomie elementu skończonego, co jest konieczne do uniknięcia blokowania objętościowego (*volumetric locking*) [64], [83]. W wariancie nieściśliwym człon W_V nie występuje, a $J = 1$ jest wymuszany przez wprowadzenie ciśnienia hydrostatycznego jako dodatkowej niewiadomej węzłowej (mnożnika Lagrange'a), co wymaga użycia elementów hybrydowych.

Warto zauważyć, że addytywny rozkład na część izochoryczną i objętościową jest fizycznie uzasadniony wyłącznie dla materiałów małościśliwych [67] (*nearly incompressible* [64]) — gdy $\kappa \gg \mu$, zmiana kształtu prawie nie wpływa na objętość i odwrotnie, więc oba rodzaje odkształceń można traktować niezależnie. Dla materiału o porównywalnych κ i μ ta niezależność zanika — np. czyste ścinanie, które z definicji jest deformacją izochoryczną, powodowałoby zauważalną zmianę objętości, co podważa sensowność osobnego traktowania obu członów.

5.7. Ogólna postać relacji konstytutywnych hipersprężystości w zagadnieniach płaskich

W wielu zastosowaniach inżynierskich trójwymiarowy opis konstytutywny można uprościć, sprowadzając go do zagadnienia płaskiego. Wyróżnia się dwa zasadnicze przypadki: płaski stan naprężenia (PSN), w którym jedna z głównych wartości naprężenia jest równa zeru, oraz płaski stan odkształcenia (PSO), w którym zerowe jest odpowiadające odkształcenie główne [64], [66], [67]. W niniejszej pracy rozpatruje się wyłącznie płaski stan naprężenia, który stanowi naturalne założenie zarówno dla klasycznych testów kalibracyjnych na cienkich próbkach — jednoosiowego rozciągania, dwuosiowego rozciągania czy czystego ścinania — jak i dla współczesnych metod pomiarowych opartych na cyfrowej korelacji obrazu (DIC, ang. *Digital Image Correlation*) [84]–[87], gdzie śledzona jest zazwyczaj deformacja elementów, w których jeden wymiar jest znacząco mniejszy od pozostałych.

5.7.1. Płaski stan naprężenia

W PSN tensor naprężenia Cauchy'ego przyjmuje postać:

$$\boldsymbol{\sigma} = \tilde{\boldsymbol{\sigma}} + 0\,\mathbf{n} \otimes \mathbf{n}, \tag{5.61}$$

gdzie $\tilde{\boldsymbol{\sigma}}$ jest płaskim symetrycznym tensorem naprężenia, a $\mathbf{n}$ oznacza wektor własny stanu naprężenia odpowiadający zerowemu naprężeniu głównemu.

W zagadnieniach płaskiego stanu naprężenia reprezentacja macierzowa tensora gradientu deformacji przyjmuje postać:

$$\mathbf{F} \to \begin{bmatrix} F_{11} & F_{12} & 0 \\ F_{21} & F_{22} & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}, \tag{5.62}$$

gdzie λ_3 jest wydłużeniem w kierunku normalnym do płaszczyzny deformacji. Wyznacznik płaskiego gradientu deformacji $\tilde{\mathbf{F}}$ oznaczamy $\tilde{J} = \det \tilde{\mathbf{F}}$, wobec czego:

$$J = \det \mathbf{F} = \tilde{J}\, \lambda_3. \tag{5.63}$$

Lewy tensor Cauchy'ego–Greena $\mathbf{B} = \mathbf{F}\mathbf{F}^T$ przyjmuje postać:

$$\mathbf{B} = \tilde{\mathbf{B}} + \lambda_3^2\, \mathbf{n} \otimes \mathbf{n}, \qquad \mathbf{B}^{-1} = \tilde{\mathbf{B}}^{-1} + \lambda_3^{-2}\, \mathbf{n} \otimes \mathbf{n}, \tag{5.64}$$

gdzie $\tilde{\mathbf{B}} = \tilde{\mathbf{F}}\tilde{\mathbf{F}}^T$. Izochoryczny lewy tensor Cauchy'ego–Greena definiujemy jako:

$$\bar{\mathbf{B}} = J^{-2/3}\, \mathbf{B}, \qquad \bar{\mathbf{B}}^{-1} = J^{2/3}\, \mathbf{B}^{-1}. \tag{5.65}$$

Dewiatorowe części tych tensorów:

$$\bar{\mathbf{B}}_D = \bar{\mathbf{B}} - \tfrac{1}{3}\,(\operatorname{tr} \bar{\mathbf{B}})\, \mathbf{I}, \qquad \bar{\mathbf{B}}_D^{-1} = \bar{\mathbf{B}}^{-1} - \tfrac{1}{3}\,(\operatorname{tr} \bar{\mathbf{B}}^{-1})\, \mathbf{I}. \tag{5.66}$$

Podstawiając (5.64) do (5.65) otrzymujemy składową $\bar{B}_{33}$ oraz ślad tensora izochorycznego:

$$\bar{B}_{33} = J^{-2/3}\, \lambda_3^2, \qquad \operatorname{tr} \bar{\mathbf{B}} = J^{-2/3} \operatorname{tr} \mathbf{B} = J^{-2/3} \Big(\operatorname{tr} \tilde{\mathbf{B}} + \lambda_3^2 \Big). \tag{5.67}$$

Stąd z (5.66) składowa dewiatorowa w kierunku normalnym wynosi:

$$(\bar{\mathbf{B}}_D)_{33} = \bar{B}_{33} - \tfrac{1}{3} \operatorname{tr} \bar{\mathbf{B}} = J^{-2/3}\, \lambda_3^2 - \tfrac{1}{3}\, J^{-2/3} \Big(\operatorname{tr} \tilde{\mathbf{B}} + \lambda_3^2 \Big) = \tfrac{1}{3}\, J^{-2/3} \Big(2\lambda_3^2 - \operatorname{tr} \tilde{\mathbf{B}} \Big). \tag{5.68}$$

Analogicznie dla tensora odwrotnego, podstawiając (5.64) do (5.65):

$$\bar{B}^{-1}_{33} = J^{2/3}\,\lambda_3^{-2}, \qquad \operatorname{tr}\bar{\mathbf{B}}^{-1} = J^{2/3}\operatorname{tr}\mathbf{B}^{-1} = J^{2/3}\Big(\operatorname{tr}\tilde{\mathbf{B}}^{-1} + \lambda_3^{-2}\Big), \tag{5.69}$$

skąd:

$$(\bar{\mathbf{B}}^{-1}_D)_{33} = \bar{B}^{-1}_{33} - \tfrac{1}{3}\operatorname{tr}\bar{\mathbf{B}}^{-1} = J^{2/3}\,\lambda_3^{-2} - \tfrac{1}{3}\,J^{2/3}\Big(\operatorname{tr}\tilde{\mathbf{B}}^{-1} + \lambda_3^{-2}\Big) = \tfrac{1}{3}\,J^{2/3}\Big(2\lambda_3^{-2} - \operatorname{tr}\tilde{\mathbf{B}}^{-1}\Big). \tag{5.70}$$

Wprowadzając następujące współczynniki:

$$\bar{\beta}_0 = \frac{\partial U}{\partial J}, \qquad \bar{\beta}_1 = \frac{2}{J}\,\frac{\partial \bar{W}}{\partial \bar{I}_1}, \qquad \bar{\beta}_{-1} = -\frac{2}{J}\,\frac{\partial \bar{W}}{\partial \bar{I}_2}, \tag{5.71}$$

relację (5.47) zapisujemy w zwartej postaci:

$$\boldsymbol{\sigma} = \bar{\beta}_0\,\mathbf{I} + \bar{\beta}_1\,\bar{\mathbf{B}}_D + \bar{\beta}_{-1}\,\bar{\mathbf{B}}^{-1}_D. \tag{5.72}$$

Podstawiając rozkłady (5.64) oraz $\mathbf{I} = \tilde{\mathbf{I}} + \mathbf{n}\otimes\mathbf{n}$ do (5.72), rozdzielamy tensor naprężenia na część płaską i składową normalną:

$$\boldsymbol{\sigma} = \tilde{\boldsymbol{\sigma}} + \sigma_{33}\,\mathbf{n}\otimes\mathbf{n}, \tag{5.73}$$

gdzie:

$$\tilde{\boldsymbol{\sigma}} = \bar{\beta}_0\,\tilde{\mathbf{I}} + \bar{\beta}_1\,(\bar{\mathbf{B}}_D)_{\|} + \bar{\beta}_{-1}\,(\bar{\mathbf{B}}^{-1}_D)_{\|} \tag{5.74}$$

jest płaskim tensorem naprężenia, zaś:

$$\sigma_{33} = \bar{\beta}_0 + \bar{\beta}_1\,(\bar{\mathbf{B}}_D)_{33} + \bar{\beta}_{-1}\,(\bar{\mathbf{B}}^{-1}_D)_{33}. \tag{5.75}$$

Dla ścisłości, $(\bar{\mathbf{B}}_D)_{\|}$ oraz $(\bar{\mathbf{B}}^{-1}_D)_{\|}$ oznaczają bloki płaskie (2×2) dewiatorów trójwymiarowych, tj.:

$$(\bar{\mathbf{B}}_D)_{\|} = J^{-2/3}\Big[\,\tilde{\mathbf{B}} - \tfrac{1}{3}\Big(\operatorname{tr}\tilde{\mathbf{B}} + \lambda_3^2\Big)\,\tilde{\mathbf{I}}\Big], \tag{5.76}$$

$$(\bar{\mathbf{B}}^{-1}_D)_{\|} = J^{2/3}\Big[\,\tilde{\mathbf{B}}^{-1} - \tfrac{1}{3}\Big(\operatorname{tr}\tilde{\mathbf{B}}^{-1} + \lambda_3^{-2}\Big)\,\tilde{\mathbf{I}}\Big], \tag{5.77}$$

co różni je od dewiatorów dwuwymiarowych tensora płaskiego.

Warunek PSN $\sigma_{33} = 0$ z wykorzystaniem (5.68) i (5.70) daje równanie na λ_3:

$$\bar{\beta}_0 + \bar{\beta}_1 \tfrac{1}{3} J^{-2/3}\Big(2\lambda_3^2 - \operatorname{tr}\tilde{\mathbf{B}}\Big) + \bar{\beta}_{-1} \tfrac{1}{3} J^{2/3}\Big(2\lambda_3^{-2} - \operatorname{tr}\tilde{\mathbf{B}}^{-1}\Big) = 0, \qquad J = \tilde{J}\,\lambda_3. \tag{5.78}$$

Rozwiązywalność analityczna tego równania zależy od przyjętej postaci funkcji JES. Stosując MES równanie (5.78) rozwiązuje się iteracyjnie (np. metodą Newtona–Raphsona) w każdym punkcie całkowania Gaussa [88].

Po wyznaczeniu λ_3 (analitycznie lub numerycznie) buduje się pełny tensor gradientu deformacji z (5.62), co pozwala obliczyć wszystkie składowe naprężenia z relacji trójwymiarowej (5.72). Można równoważnie posłużyć się bezpośrednio postacią płaską (5.74) — oba zapisy dają identyczny wynik, ponieważ $\sigma_{33} = 0$ jest już zagwarantowane przez dobór λ_3.

Analogiczną równoważność można wykazać dla tensora naprężenia Pioli–Kirchhoffa I rodzaju $\mathbf{P} = J\,\boldsymbol{\sigma}\,\mathbf{F}^{-T}$. Korzystając (5.62), (5.63), płaski tensor tego naprężenia możemy zapisać jako:

$$\tilde{\mathbf{S}} = \lambda_3\,\tilde{J}\,\tilde{\boldsymbol{\sigma}}\,\tilde{\mathbf{F}}^{-T}. \tag{5.79}$$

Dla materiału nieściśliwego ($J = 1$) zachodzi $\lambda_3 = 1/\tilde{J}$, a warunek (5.78) jest spełniony tożsamościowo — rolę $\bar{\beta}_0$ przejmuje mnożnik Lagrange'a $(-p)$, eliminowany algebraicznie z równania $\sigma_{33} = 0$. Relacja konstytutywna przyjmuje wówczas postać [72]:

$$\boldsymbol{\sigma} = -p\,\mathbf{I} + \bar{\beta}_1\,\bar{\mathbf{B}} + \bar{\beta}_{-1}\,\bar{\mathbf{B}}^{-1}, \qquad \bar{\beta}_1 = 2\,\frac{\partial U}{\partial \bar{I}_1}, \quad \bar{\beta}_{-1} = -2\,\frac{\partial U}{\partial \bar{I}_2}, \tag{5.80}$$

gdzie ciśnienie wyznacza się z warunku $\sigma_{33} = 0$:

$$p = \bar{\beta}_1\,\lambda_3^2 + \bar{\beta}_{-1}\,\lambda_3^{-2} = \bar{\beta}_1\,\tilde{J}^{-2} + \bar{\beta}_{-1}\,\tilde{J}^2. \tag{5.81}$$

Po podstawieniu (5.81) do (5.80) oraz wykorzystaniu rozkładów $\bar{\mathbf{B}} = \tilde{\mathbf{B}} + \tilde{J}^{-2}\,\mathbf{n}\otimes\mathbf{n}$, $\bar{\mathbf{B}}^{-1} = \tilde{\mathbf{B}}^{-1} + \tilde{J}^2\,\mathbf{n}\otimes\mathbf{n}$ otrzymuje się płaską relację konstytutywną PSN [72]:

$$\tilde{\boldsymbol{\sigma}} = -\Big(\bar{\beta}_1\,\tilde{J}^{-2} + \bar{\beta}_{-1}\,\tilde{J}^2\Big)\,\tilde{\mathbf{I}} + \bar{\beta}_1\,\tilde{\mathbf{B}} + \bar{\beta}_{-1}\,\tilde{\mathbf{B}}^{-1}. \tag{5.82}$$

Po przekształceniach, zastosowaniu twierdzenia Cayley'a–Hamiltona dla tensorów płaskich $\tilde{\mathbf{B}}^{-1} = -\tilde{J}^{-2}\big(\tilde{\mathbf{B}} - \tilde{I}_1\,\tilde{\mathbf{I}}\big)$, otrzymujemy [72]:

$$\tilde{\boldsymbol{\sigma}} = -\Big(\bar{\beta}_1\,\tilde{J}^{-2} + \bar{\beta}_{-1}\big(\tilde{J}^2 - \tilde{I}_1\big)\Big)\,\tilde{\mathbf{I}} + \big(\bar{\beta}_1 - \bar{\beta}_{-1}\,\tilde{J}^{-2}\big)\,\tilde{\mathbf{B}}. \tag{5.83}$$

Wykorzystanie więzu nieściśliwości $\lambda_3 = (\lambda_1\lambda_2)^{-1}$ sprawia, że w PSN funkcję energii można uzależnić wyłącznie od dwóch wartości własnych [72]:

$$\overline{U}\big(\bar{\lambda}_1, \bar{\lambda}_2\big) = \widetilde{W}\big(\bar{\lambda}_1, \bar{\lambda}_2, \bar{\lambda}_1^{-1}\bar{\lambda}_2^{-1}\big)\,. \tag{5.84}$$

Główne naprężenia Pioli-Kirchhoffa I rodzaju przyjmują wtedy postać:

$$S_1 = \frac{\partial \overline{U}}{\partial \bar{\lambda}_1}, \qquad S_2 = \frac{\partial \overline{U}}{\partial \bar{\lambda}_2}. \tag{5.85}$$

Funkcja $\overline{U}(\bar{\lambda}_1, \bar{\lambda}_2)$ jest ponadto ściśle wypukła wtedy, gdy macierz:

$$\begin{bmatrix} \dfrac{\partial^2 \overline{U}}{\partial \bar{\lambda}_1^2} & \dfrac{\partial^2 \overline{U}}{\partial \bar{\lambda}_1\,\partial \bar{\lambda}_2} \\[2ex] \dfrac{\partial^2 \overline{U}}{\partial \bar{\lambda}_1\,\partial \bar{\lambda}_2} & \dfrac{\partial^2 \overline{U}}{\partial \bar{\lambda}_2^2} \end{bmatrix} \tag{5.86}$$

pozostaje dodatnio określona [72], [89]. Mniej restrykcyjnym warunkiem jest stabilność w sensie Druckera — materiał nieściśliwy uznaje się za stabilny, jeżeli poniższa macierz funkcyjna jest dodatnio określona [72], [90]:

$$\begin{bmatrix} \bar{\lambda}_1 \dfrac{\partial \overline{U}}{\partial \bar{\lambda}_1} + \bar{\lambda}_1^2 \dfrac{\partial^2 \overline{U}}{\partial \bar{\lambda}_1^2} & \bar{\lambda}_1\bar{\lambda}_2 \dfrac{\partial^2 \overline{U}}{\partial \bar{\lambda}_1\,\partial \bar{\lambda}_2} \\[2ex] \bar{\lambda}_1\bar{\lambda}_2 \dfrac{\partial^2 \overline{U}}{\partial \bar{\lambda}_1\,\partial \bar{\lambda}_2} & \bar{\lambda}_2 \dfrac{\partial \overline{U}}{\partial \bar{\lambda}_2} + \bar{\lambda}_2^2 \dfrac{\partial^2 \overline{U}}{\partial \bar{\lambda}_2^2} \end{bmatrix}. \tag{5.87}$$

5.7.2. Płaski stan odkształcenia

W płaskim stanie odkształcenia (PSO) izochoryczny lewy tensor deformacji Cauchy'ego–Greena przyjmuje postać

$$\bar{\mathbf{B}} = \tilde{\mathbf{B}} + \mathbf{n} \otimes \mathbf{n}, \tag{5.88}$$

gdzie $\tilde{\mathbf{B}}$ jest płaską (symetryczną) częścią tensora, a $\mathbf{n}$ oznacza wektor normalny do płaszczyzny odkształcenia, odpowiadający jednostkowemu wydłużeniu w tym kierunku. Dla materiałów nieściśliwych w warunkach PSO trzecia izochoryczna wartość własna wydłużenia wynosi $\bar{\lambda}_3 = 1$, a z warunku nieściśliwości $\bar{\lambda}_1\bar{\lambda}_2 = 1$ wynika, że oba izochoryczne niezmienniki są sobie równe:

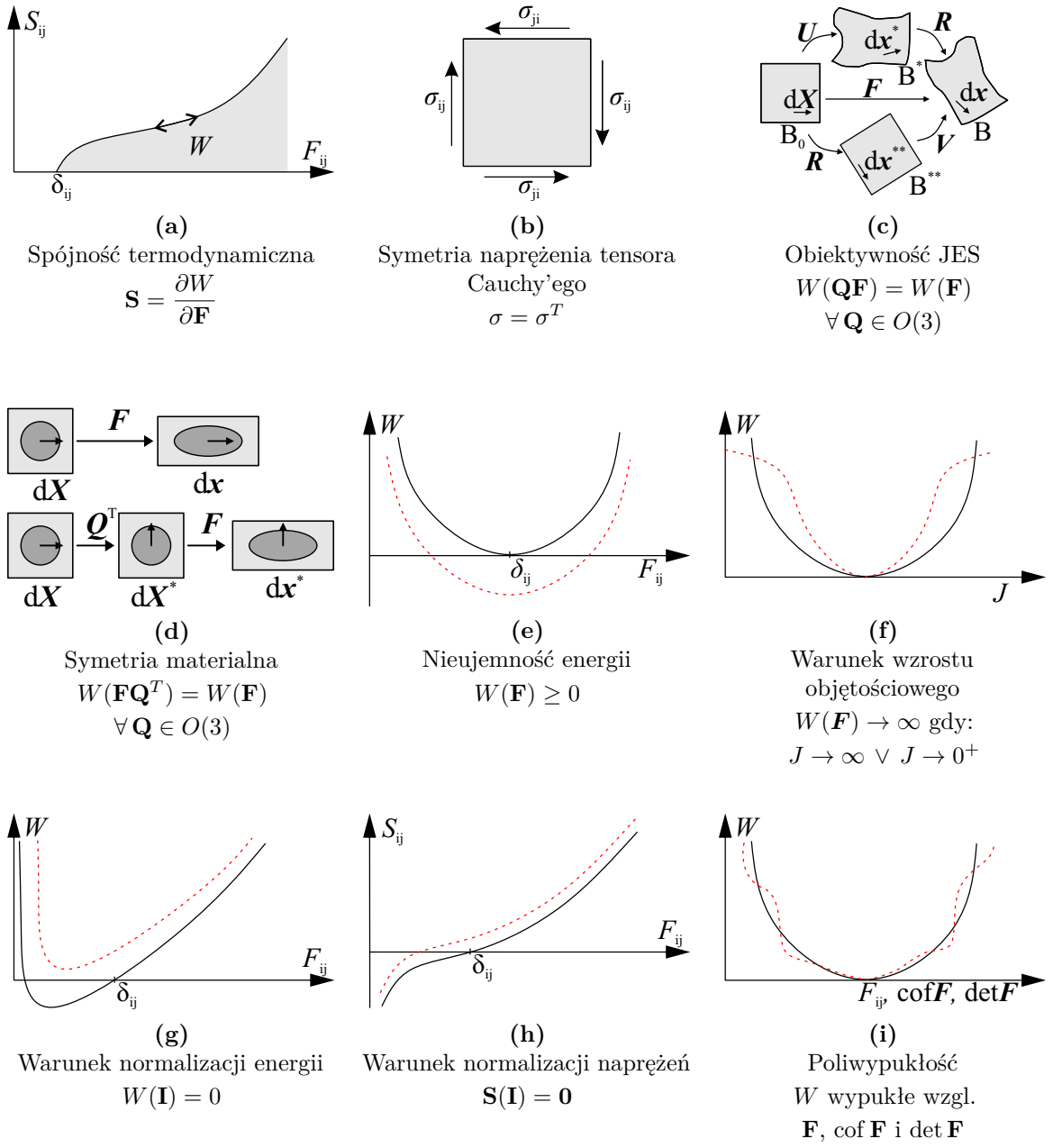
$$\bar{I}_1 = \bar{I}_2 = \bar{\lambda}_1^2 + \bar{\lambda}_1^{-2} + 1. \tag{5.89}$$

Funkcja energii odkształcenia redukuje się zatem do zależności od jednej wartości własnej.

W niniejszej pracy przypadek PSO nie jest dalej rozwijany, ponieważ pomiary z cyfrowej korelacji obrazu na cienkich próbkach odpowiadają warunkom PSN.

5.8. Wymagania dla modelowania relacji konstytutywnych stawiane przez Mechanikę Ośrodków Ciągłych

Celem modelowania konstytutywnego w teorii sprężystości jest znalezienie związku pomiędzy odkształceniem a wywoływanym przez nie naprężeniem w punkcie materialnym przy spełnieniu szeregu wymagań stawianych potencjałowi sprężystemu i naprężeniom [91]. W niniejszym rozdziale zostaną omówione te wymagania oraz przedstawione zostaną metody ich spełnienia.



(a) Spójność termodynamiczna $\mathbf{S} = \dfrac{\partial W}{\partial \mathbf{F}}$

(b) Symetria naprężenia tensora Cauchy'ego $\sigma = \sigma^T$

(c) Obiektywność JES $W(\mathbf{QF}) = W(\mathbf{F})$ $\forall\, \mathbf{Q} \in O(3)$

(d) Symetria materialna $W(\mathbf{FQ}^T) = W(\mathbf{F})$ $\forall\, \mathbf{Q} \in O(3)$

(e) Nieujemność energii $W(\mathbf{F}) \geq 0$

(f) Warunek wzrostu objętościowego $W(\boldsymbol{F}) \to \infty$ gdy: $J \to \infty \ \vee \ J \to 0^+$

(g) Warunek normalizacji energii $W(\mathbf{I}) = 0$

(h) Warunek normalizacji naprężeń $\mathbf{S}(\mathbf{I}) = \mathbf{0}$

(i) Poliwypukłość W wypukłe wzgl. $\mathbf{F}$, $\operatorname{cof}\mathbf{F}$ i $\det\mathbf{F}$

Rysunek 5.3. Schematyczne przedstawienie typowych warunków dla potencjału sprężystego i naprężeń [91]

5.8.1. Spójność termodynamiczna

Relacje konstytutywne hipersprężystości muszą być spójne z zasadami termodynamiki. W procesach izotermicznych i adiabatycznych energia wewnętrzna $\dot{U}$ jest równa gęstości energii swobodnej Helmholtza, a z całkowego bilansu energii mechanicznej oraz zasad zachowania pędu, momentu pędu i masy otrzymuje się następujące zależności lokalne [67],

[72]:

$$\dot{U} = \mathrm{tr}(\boldsymbol{\sigma}\mathbf{D}), \qquad \rho_0\dot{U} = \mathrm{tr}(\mathbf{S}^T\dot{\mathbf{F}}). \tag{5.90}$$

Wprowadzając pojęcie jednostkowej energii sprężystości (JES) W i stosując lokalną zasadę zachowania masy $\rho_0 = \rho J$, zależność (5.90) przyjmuje postać:

$$\dot{W} = \frac{1}{J}\mathrm{tr}(\boldsymbol{\sigma}\mathbf{D}) = \mathrm{tr}(\mathbf{S}^T\dot{\mathbf{F}}). \tag{5.91}$$

Z lokalnego sformułowania zasady zachowania energii mechanicznej (5.91), dla dostatecznie regularnej funkcji JES materiału jednorodnego, wynika, że funkcja W jest potencjałem dla naprężeń, tzn.:

$$\mathbf{S}(\mathbf{F}) = \frac{\partial W(\mathbf{F})}{\partial \mathbf{F}}, \qquad \mathbf{T} = 2\frac{\partial W}{\partial \mathbf{C}}. \tag{5.92}$$

Istnienie potencjału (5.92) gwarantuje, że na zamkniętym cyklu deformacji dyssypacja energii jest zerowa:

$$\oint \mathrm{tr}(\mathbf{T}\dot{\mathbf{E}})\, dt = 0, \tag{5.93}$$

co stanowi fundamentalną różnicę między hipersprężystością (sprężystością w sensie Greena) a hyposprężystością, gdzie warunek (5.93) w ogólności nie jest spełniony [67], [72].

5.8.2. Symetria tensora naprężeń Cauchy'ego

Tensor naprężeń Cauchy'ego $\boldsymbol{\sigma}$ jest symetryczny, co wynika bezpośrednio z zasady zachowania momentu pędu. Z zasad zachowania pędu i momentu pędu w konfiguracji aktualnej otrzymuje się następujące lokalne równania równowagi [66], [67]:

$$\mathrm{div}\,\boldsymbol{\sigma} + \boldsymbol{f} = \rho\,\ddot{\boldsymbol{u}}\,, \qquad \boldsymbol{\sigma} = \boldsymbol{\sigma}^T\,, \tag{5.94}$$

gdzie $\boldsymbol{f}$ oznacza siły objętościowe, ρ – gęstość ciała w konfiguracji aktualnej, a $\ddot{\boldsymbol{u}}$ – przyspieszenie.

5.8.3. Obiektywność funkcji JES i symetria materialna

Wymóg obiektywności oraz izotropii materiału narzuca na funkcję JES następujący warunek:

$$W(\mathbf{E}) = W\left(\mathbf{Q}\mathbf{E}\mathbf{Q}^T\right),\ \forall \mathbf{Q} \in O(3), \tag{5.95}$$

w którym tensory $\mathbf{Q}$ są elementami grupy przekształceń Galileusza, obejmującej obroty i odbicia lustrzane $O(3)$ w trójwymiarowej przestrzeni euklidesowej. Warunek ten jest równoważny rozdzielnemu spełnieniu obiektywności $W(\mathbf{F}) = W(\mathbf{Q}\mathbf{F})$ oraz symetrii materialnej $W(\mathbf{F}) = W(\mathbf{F}\mathbf{Q}^T)$ [68]–[70], [73], [74]. Na mocy warunku 5.95 oraz twierdzenia o skalarnej izotropowej funkcji tensorowej funkcja JES zależy od trzech niezależnych niezmienników tensora $\mathbf{E}$, np.:

$$W(\mathbf{E}) = \breve{W}(tr\mathbf{E},\ tr\mathbf{E}^2,\ tr\mathbf{E}^3) = \breve{W}(N_i);\ i = 1, 2, 3. \tag{5.96}$$

Podstawienie powyższego do 5.34 prowadzi do związku konstytutywnego hipersprężystości materiału izotropowego w opisie materialnym [68]–[70], [74], [76].

5.8.4. Warunek wzrostu

Dodatkowo, rozważa się różne obligatoryjne warunki, z których najczęściej spotykanym jest warunek wzrostu objętościowego:

$$W(\mathbf{F}) \to \infty \quad \text{w miarę jak} \quad (J \to 0^+ \quad \vee \quad J \to \infty), \tag{5.97}$$

co wynika z obserwacji, że ciało materialne nie może zostać skompresowane do objętości równej zero ani rozszerzone do nieskończonej objętości [64]. Szczególnie istotny jest przypadek znacznej kompresji objętościowej, ponieważ może on występować w szerokim zakresie praktycznych zastosowań inżynierskich.

5.8.5. Warunek normalizacji i nieujemności funkcji jednostkowej energii sprężystości oraz warunek normalizacji naprężeń

W niezdeformowanej konfiguracji, tj. $\mathbf{F} = \mathbf{I}$, powinny być spełnione warunki **normalizacji** [64], [91]:

$$W(\mathbf{F} = \mathbf{I}) = 0 \quad \text{oraz} \quad \mathbf{S}(\mathbf{F} = \mathbf{I}) = \mathbf{0} \tag{5.98}$$

dla obu: funkji jedostkowej energii sprężystości i naprężeń. Oprócz normalizacji, energia swobodna powinna wzrastać w każdym przypadku, gdy pojawia się deformacja. Zatem, oprócz równania 5.98_1, **nieujemność energii odkształcenia**, tj. $W(\mathbf{F}) \geq 0$ jest wymagana.

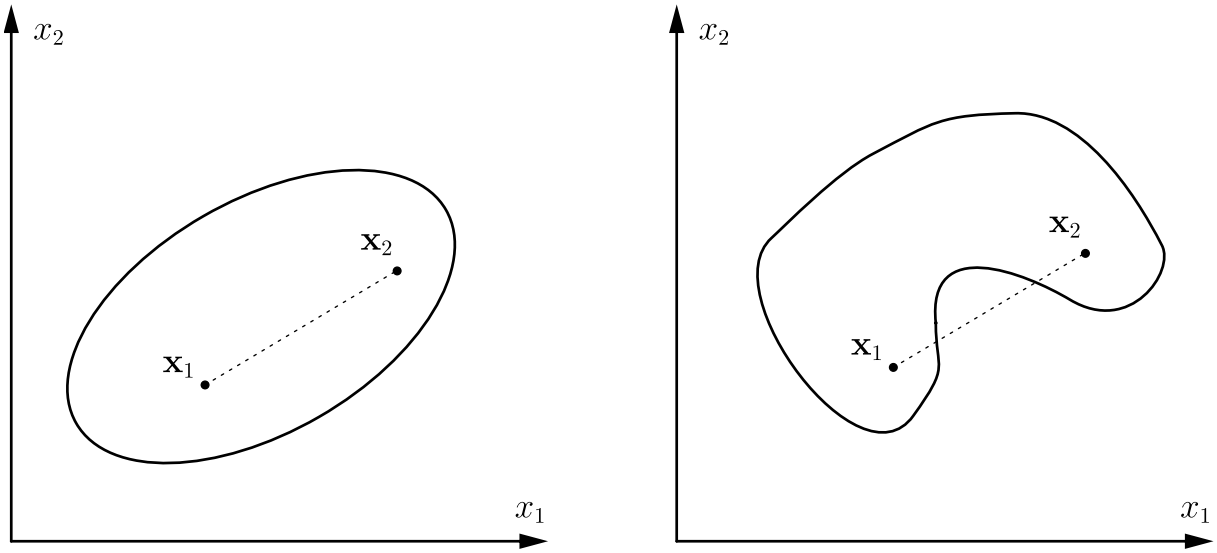
5.9. Poliwypukłość

5.9.1. Wypukłość

Funkcje *ściśle wypukłe* mają unikalne (globalne) minimum. Powstaje pytanie czy są one również użyteczne z fizycznego punktu widzenia? Aby przygotować odpowiedź, rozważmy dwuwymiarowy zbiór K. Zbiór ten jest wypukły, gdy

$$\lambda\,\mathbf{x}_1 + (1-\lambda)\,\mathbf{x}_2 \;\in\; K, \qquad \forall\,\mathbf{x}_1, \mathbf{x}_2 \in K, \quad \lambda \in [0;1]. \tag{5.99}$$

W ujęciu geometrycznym definicja ta stwierdza, że odcinek łączący $\mathbf{x}_1$ i $\mathbf{x}_2$ zawiera się w K. W przypadku zbioru wypukłego jednowymiarowego lub dziewięciowymiarowego punkty w (5.99) należy odpowiednio zastąpić skalarami oraz tensorami drugiego rzędu.



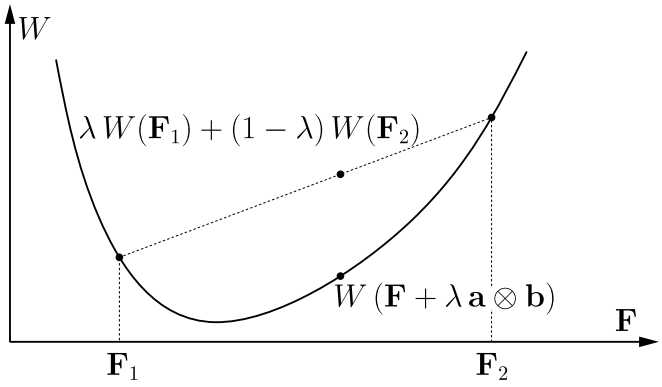
(a) Zbiór wypukły **(b)** Zbiór niewypukły

Rysunek 5.4. Porównanie zbioru wypukłego i niewypukłego: (a) i (b)

W przypadku skalarnej wypukłej funkcji tensorowej, np. funkcji JES W w zależności od gradientu deformacji $\mathbf{F}$, zdefiniowanego na dziewięciowymiarowym zbiorze wypukłym

$\mathbb{R}^{3\times3}$, zachodzi, por. rys. 5.5:

$$W(\lambda\mathbf{F}_1 + (1-\lambda)\mathbf{F}_2) \;\leq\; \lambda\,\psi(\mathbf{F}_1) \;+\; (1-\lambda)\,\psi(\mathbf{F}_2) \quad \forall\,\mathbf{F}_1,\mathbf{F}_2 \in \mathbb{R}^{3\times3},\; \lambda\in[0,1]. \tag{5.100}$$

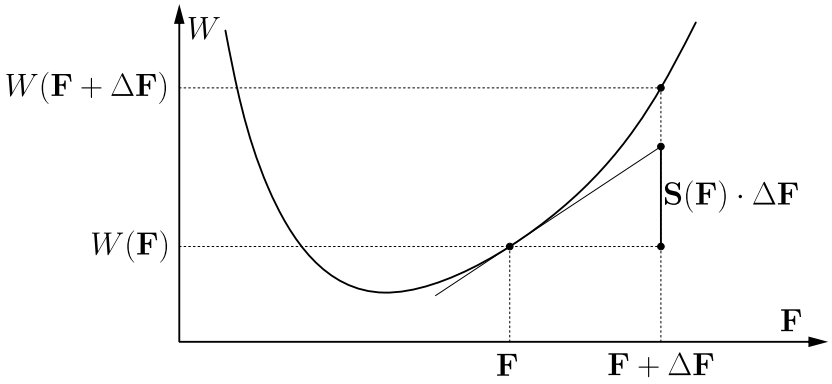


Rysunek 5.5. Wypukłość funkcji $W(\mathbf{F})$

Jeżeli funkcja $W(\mathbf{F})$ jest różniczkowalna, warunek wypukłości przyjmuje postać, por. rys. 5.6:

$$W(\mathbf{F}) \;+\; \mathbf{S}(\mathbf{F})\cdot\Delta\mathbf{F} \;\leq\; W(\mathbf{F}+\Delta\mathbf{F}) \quad \forall\,\mathbf{F},\Delta\mathbf{F}\in\mathbb{R}^{3\times3}, \tag{5.101}$$

gdzie: $\mathbf{S}(\mathbf{F}) = \frac{\partial W}{\partial \mathbf{F}}, \mathbf{F} = \mathbf{F}_1, \Delta\mathbf{F} = \mathbf{F}_2 - \mathbf{F}_1$. Dla funkcji ściśle wypukłych w równaniu (5.101) zachodzi znak "$<$" zamiast "$\leq$" przy $\mathbf{F}_1 \neq \mathbf{F}_2$ i $\lambda\in[0,1]$.



Rysunek 5.6. Wypukłość różniczkowalnej funkcji $W(\mathbf{F})$

Alternatywnie (warunek monotoniczności) można zapisać

$$(\mathbf{S}(\mathbf{F}_2) - \mathbf{S}(\mathbf{F}_1)) \cdot (\mathbf{F}_2 - \mathbf{F}_1) \begin{cases} \geq 0, & \text{(wypukłość)} \\ > 0, & \text{(ścisła wypukłość)} \end{cases} \qquad \forall\, \mathbf{F}_1, \mathbf{F}_2 \in \mathbb{R}^{3\times 3}. \tag{5.102}$$

Jeżeli W jest dwukrotnie różniczkowalna, równoważny warunek wypukłości zapisujemy jako

$$\Delta\mathbf{F} \cdot \mathbf{A} \cdot \Delta\mathbf{F} \begin{cases} \geq 0, & \text{(wypukłość)} \\ > 0, & \text{(ścisła wypukłość)} \end{cases} \qquad \forall\, \mathbf{F}, \Delta\mathbf{F} \in \mathbb{R}^{3\times 3}, \quad \mathbf{A} = \frac{\partial^2 W(\mathbf{F})}{\partial \mathbf{F}\, \partial \mathbf{F}}. \tag{5.103}$$

co stanowi wymóg dodatniej (pół)określoności tensora styczności nominalnej $\mathbf{A}$. Niemniej jednak warunku wypukłości nie można stosować, ponieważ własność ta stoi w sprzeczności z najbardziej bezpośrednimi wymaganiami fizycznymi (por. np.[61]).

Zbiór $\{\mathbf{F} \mid \det \mathbf{F} > 0\}$, przy $\{\mathbf{F} \in \mathbb{R}^{3\times 3}_{+}\}$ nie jest wypukły. Aby to zilustrować, rozważa się prosty przykład. Definiuje się dwa gradienty odkształcenia:

$$\mathbf{F}_1 = \operatorname{diag}(1,1,1), \mathbf{F}_2 = \operatorname{diag}(-1,-1,1), \qquad \text{przy } \det \mathbf{F}_1 = 1, \det \mathbf{F}_2 = 1. \tag{5.104}$$

Ich wypukła kombinacja liniowa dla $\lambda = \frac{1}{2}$ daje

$$\mathbf{F} = \tfrac{1}{2}\,\mathbf{F}_1 + \tfrac{1}{2}\,\mathbf{F}_2 = \operatorname{diag}(0,0,1), \qquad \det \mathbf{F} = 0. \tag{5.105}$$

Stąd wprost widać, że $\det \mathbf{F}$ dla kombinacji wypukłej może spaść do zera, co przeczy wypukłości zbioru $\{\mathbf{F} \mid \det \mathbf{F} > 0\}$.

Nieadekwatność założenia wypukłości potencjału sprężystego można zaobserwować w zachowaniu jednorodnej, izotropowej, małościśliwej gumowej membrany [92]. Załóżmy poniższą postać deformacji:

$$\mathbf{F} \to \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.106}$$

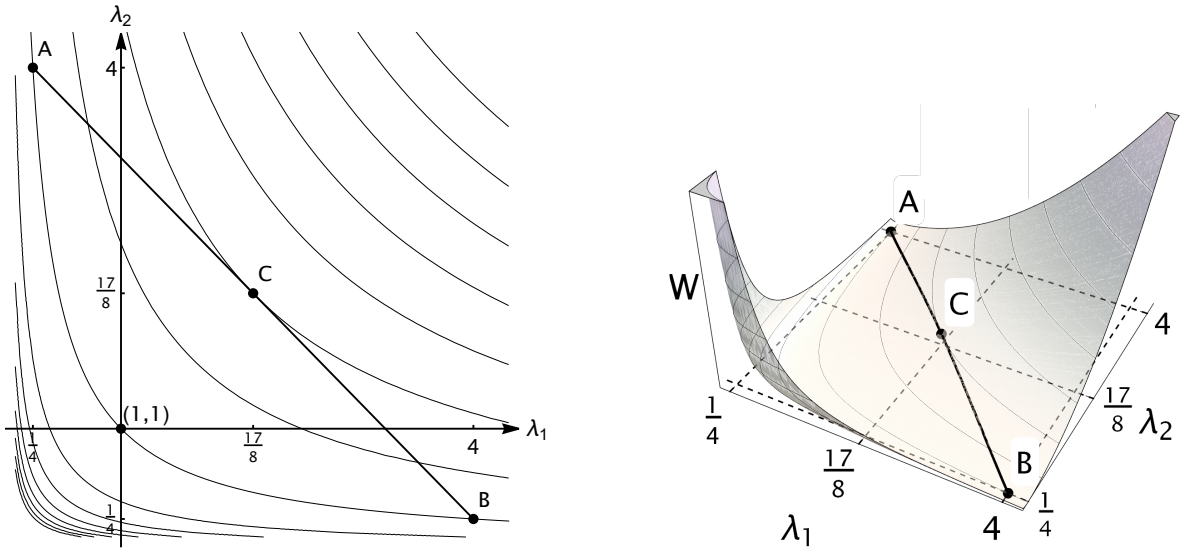
Rozważając funkcję jednostkowej energii sprężystości jako:

$$W(\mathbf{F}) = \tilde{W}(\det \mathbf{F}) = \left[(\det \mathbf{F})^2 + \frac{1}{(\det \mathbf{F})^2} - 2\right]^{\alpha} = \left(\lambda_1^2\lambda_2^2 + \frac{1}{\lambda_1^2\lambda_2^2} - 2\right)^{\alpha}, \quad \alpha \geq 1, \tag{5.107}$$

otrzymujemy:

$$W(\mathbf{F})\Big|_{\det \mathbf{F}=1} = 0, \quad \lim_{\det \mathbf{F}\to 0^+} W(\mathbf{F}) = +\infty, \quad \lim_{\det \mathbf{F}\to +\infty} W(\mathbf{F}) = +\infty. \tag{5.108}$$

Guma jest materiałem prawie nieściśliwym, a w związku z tym gromadzi dużo energii



(a) wykres warstwicowy **(b)** wykres trójwymiarowy

Rysunek 5.7. Niewypukła funkcja JES wg propozycji z równania 5.107 przy $\alpha = 1$

sprężystej, gdy iloczyn $\lambda_1\lambda_2$ znacznie odbiega od jedynki. Kontury przedstawiające równą energią są w kształcie banana (por. rys. 5.7). Sprawdzając obie strony nierówności (5.101) podstawiamy wartości jednostkowej energii sprężystości (5.107) dla $\alpha = 1$ w punktach (A), (B) i (C) z rys. 5.7. Otrzymujemy wówczas:

$$\begin{aligned} &W(\mathbf{F}_C) = W\Big(\mathrm{diag}\left(\tfrac{15}{8},\, \tfrac{15}{8},\, 1\right)\Big) = 10.44, \\ &\tfrac{1}{2}\left[W(\mathbf{F}_A) + W(\mathbf{F}_B)\right] = \tfrac{1}{2}\Big[W\Big(\mathrm{diag}\left(\tfrac{1}{4},\, 4,\, 1\right)\Big] + W\Big(\mathrm{diag}\Big(4,\, \tfrac{1}{4},\, 1\Big)\Big)\Big] = 0, \\ &W(\mathbf{F}_C) = 10.44 > \tfrac{1}{2}\left[W(\mathbf{F}_A) + W(\mathbf{F}_B)\right] = 0, \end{aligned} \tag{5.109}$$

co narusza warunek wypukłości. Notabene w literaturze równanie (5.107) dla $\alpha = 1$ powszechnie stosowane jest jako forma penalizacji objętościowej [60].

W literaturze podkreśla się, że połączenie zasad obiektywności JES i symetrii materialnej z wymogiem wypukłości JES prowadzi do szeregu nierówności ograniczających możliwe wartości własne naprężeń Kirchhoffa. Z jednej strony warunek niezmienniczości względem obrotów wymaga, aby energia nie zmieniała w zależności od konfiguracji (5.95), a z drugiej strony wypukłość funkcji energii nakłada restrykcje na liniową aproksymację tej energii (5.101), co przekłada się na niemożność wystąpienia pewnych stanów ściskania w dwóch różnych kierunkach głównych. W praktyce oznacza to, że przy założeniu jednoczesnej wypukłości i pełnej obiektywności JES można uzyskać fizycznie nieakceptowalne zachowania materiału, szczególnie w stanach czystego ściskania.

Ponadto ściśle wypukłe funkcje JES posiadają co najwyżej jedno minimum, co w rezultacie uniemożliwia wystąpienie zjawiska utraty stateczności.

5.9.2. Quasi-wypukłość

Niemożliwe jest uwolnienie energii z jednorodnego ciała przez proces izotermiczny, o ile ciało jest unieruchomione na brzegach [52]. Dla materiałów hipersprężystych ten warunek można zapisać w postaci:

$$\int_{t_0}^{t}\int_{B_0} \mathbf{S}\cdot\dot{\mathbf{F}}\,\mathrm{d}V\,\mathrm{d}t \;=\; \int_{B_0} W(\mathbf{F},t)\,\mathrm{d}V \;-\; \int_{B_0} W(\,\overline{\mathbf{F}},\,t_0\,)\,\mathrm{d}V \;\geq\; 0, \tag{5.110}$$

przy założeniu, że w chwili początkowej t_0 zachodzi $\mathbf{F}=\overline{\mathbf{F}}$ na $B_0, W(\mathbf{F},t_0)=W\left(\overline{\mathbf{F}}\right)$ oraz zachodzą jednorodne warunki brzegowe typu Dirichleta $\mathbf{x} \;=\; \overline{\mathbf{F}}\,\mathbf{X}$ na ∂B_0.

Dla odpowiednio regularnego pola wektorowego $\mathbf{v}:\mathbf{v}\in C_0^{\infty}(\Omega)$ zadanego na zbiorze $\boldsymbol{\Omega}$ o objętości V i gradientu deformacji przyjmującego postać:

$$\overline{\mathbf{F}}=\mathbf{F}+\mathrm{GRAD}\,\mathbf{v}(\mathbf{X}),\quad \overline{\mathbf{F}}\;\in\;\mathrm{Lin}^{+} \tag{5.111}$$

po podstawieniu (5.111) do (5.110) $\forall\mathbf{X}\in\Omega$ zachodzi warunek quasi-wypukłości funkcji JES [61], [71], [93], [94]:

$$\frac{1}{V}\int_{\Omega} W\left(\mathbf{F}+\mathrm{GRAD}\,\mathbf{v}(\mathbf{F})\right)\,\mathrm{d}\mathbf{X} \;\geq\; W(\mathbf{F}). \tag{5.112}$$

Warunek quasi-wypukłości musi być spełniony dla dowolnego pola wektorowego $\mathbf{v}$, nie jest wymaganiem lokalnym, a dotyczy ograniczonego zbioru punktów Ω, stąd warunek ten jest dość skomplikowany do weryfikacji[95].

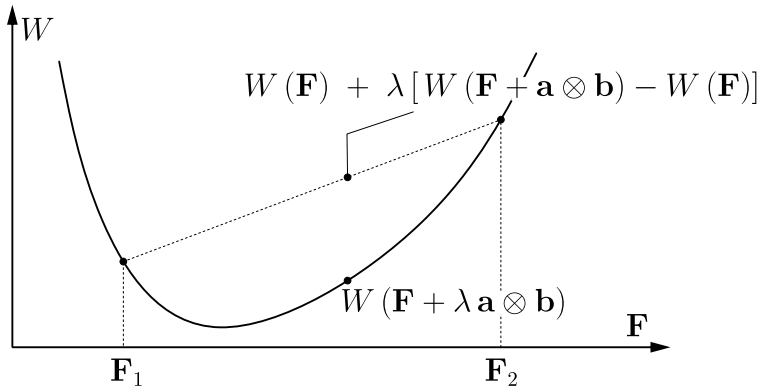
5.9.3. Wypukłość pierwszego rzędu

(Ścisła) wypukłość pierwszego rzędu ma tę samą postać, co ogólna definicja wypukłości (5.101) z tą różnicą, że:

$$\mathbf{F}_2 = \mathbf{F}_1 + \mathbf{a} \otimes \mathbf{b}, \quad \forall\, \mathbf{F}_1 \in \mathbb{R}^{3\times 3}, \quad \forall\, \mathbf{a}, \mathbf{b} \in \mathbb{R}^3 \setminus \{\mathbf{0}\}. \tag{5.113}$$

Wstawiając (5.113) do (5.100) otrzymujemy po przekształceniach otrzymujemy warunek (ścisłej) wypukłości pierwszego rzędu względem $\mathbf{a}$ dla każdego $\mathbf{F}$ i każdego wektora $\mathbf{b}$, por. rys. 5.8:

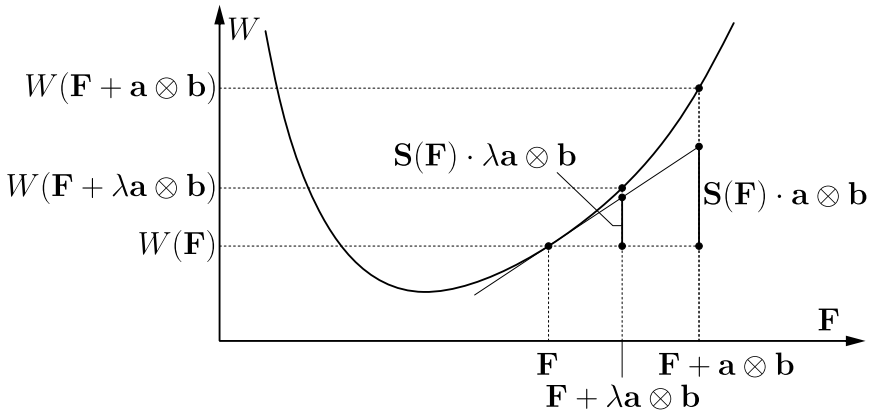
$$W\left(\mathbf{F} + \lambda\, \mathbf{a} \otimes \mathbf{b}\right)\ (<) \leq\ W\left(\mathbf{F}\right)\ +\ \lambda\left[\, W\left(\mathbf{F} + \mathbf{a} \otimes \mathbf{b}\right) - W\left(\mathbf{F}\right)\right], \quad \lambda \in [0;1] \tag{5.114}$$



Rysunek 5.8. Wypukłość rzędu pierwszego dwukrotnie różniczkowalnej funkcji $W(\mathbf{F})$

Wstawiając (5.113) do (5.101) otrzymujemy warunek wypukłości pierwszego rzędu dla funkcji W dwukrotnie różniczkowalnej, por. rys. 5.9:

$$W(\mathbf{F}) + \mathbf{S}(\mathbf{F}) \cdot (\mathbf{a} \otimes \mathbf{b})\ (<)\ \leq\ W(\mathbf{F} + \mathbf{a} \otimes \mathbf{b})\,. \tag{5.115}$$



Rysunek 5.9. Wypukłość rzędu pierwszego różniczkowalnej funkcji $W(\mathbf{F})$

Stąd tensor $\mathbf{S}$ musi być funkcją monotonicznie rosnącą względem wszystkich przyrostów rzędu pierwszego $\mathbf{a} \otimes \mathbf{b}$. Wreszcie, analogicznie do (5.103), definicję dla funkcji dwukrotnie różniczkowalnych $W(\mathbf{F})$ zapisuje się jako:

$$(\mathbf{a} \otimes \mathbf{b}) \cdot \mathbf{A}(\mathbf{F}) \cdot (\mathbf{a} \otimes \mathbf{b}) \; (>) \geq 0, \qquad \mathbf{A}(\mathbf{F}) = \frac{\partial^2 W(\mathbf{F})}{\partial \mathbf{F}^2}, \tag{5.116}$$

którą nazywa się warunkiem Legendre'a-Hamarada. Jeżeli w (5.116) występuje silna nierówność to funkcja $W(\mathbf{F})$ spełnia warunek silnej eliptyczności, który jest niezbędny do zdefiniowania dodatnio określonego tensora akustycznego, a tym samym dla istnienia rzeczywistych prędkości fal sprężystych.

W praktyce pełne wykazanie poliwypukłości analitycznie bywa trudne, dlatego stosuje się testy numeryczne sprawdzające konieczne warunki wypukłości [91], [96], [97]. Jedną z metod jest weryfikacja tzw. warunku *rank-one convexity* (wypukłości 1-rzędowej), który jest implikowany przez poliwypukłość. Polega to na sprawdzeniu, czy dla dowolnych dwóch stanów deformacji $\mathbf{F}_1, \mathbf{F}_2$ połączonych deformacją rzędu 1 (tzn. $\mathbf{F}_2 = \mathbf{F}_1 + \mathbf{a} \otimes \mathbf{b}$) energia spełnia nierówność wypukłości:

$$W(\alpha \mathbf{F}_1 + (1-\alpha)\mathbf{F}_2) \;\; \leq \;\; \alpha\, W(\mathbf{F}_1) + (1-\alpha)\, W(\mathbf{F}_2) \quad \forall\, \alpha \in [0;1]. \tag{5.117}$$

W praktyce zamiast dowolnych deformacji wybiera się reprezentatywne ścieżki odkształcenia, które najczęściej ujawniają brak poliwypukłości i W bada się wzdłuż takich ścieżek. Przykładowo, testuje się czyste ścinanie, proste ścinanie czy jednoosiowe lub dwuoosiowe rozciąganie. W jednoparametrycznych deformacjach W jako funkcja parametru deformacji

powinna być wypukła (jej druga pochodna względem tego parametru ≥ 0). Jeśli znajdzie się kierunek, w którym W jest niewypukła (tj. istnieje zakres parametru, gdzie druga pochodna staje się ujemna lub funkcja ma kształt wklęsły), oznacza to naruszenie poliwypukłości.

5.9.4. Poliwypukłość

Szeroka klasa zagadnień brzegowych w teorii hipersprężystości, sformułowanych w języku analizy funkcjonalnej, dla której udowodniono twierdzenie o istnieniu rozwiązania, opiera się na poliwypukłych potencjałach sprężystości [98]. Funkcja jednostkowa ES $W(\mathbf{F})$ jest poliwypukła, jeżeli wypukła jest jej rozszerzona postać, określona jako:

$$W(\mathbf{F}) = \mathcal{W}(\mathbf{F}, \operatorname{Cof}\mathbf{F}, \det\mathbf{F}), \tag{5.118}$$

gdzie dziedziną tej funkcji jest dziewiętnastowymiarowy zbiór $Lin \times Lin \times (0, \infty)$ [99], [100]. Rozszerzenie dziedziny potencjału sprężystości jest niezbędne, ponieważ dla funkcji $W(\cdot, F)$ tensor $F \in Lin^+$ należy do niewypukłego zbioru Lin^+. Z perspektywy matematycznej teorii hipersprężystości analiza poliwypukłości funkcji JES materiałów ściśliwych okazuje się wygodniejsza w postaci (5.36) niż w przypadku funkcji JES (5.51) lub (5.50) w dziedzinie niezmienników deformacji izochorycznej [55], [57], [67]. Niezmienniki deformacji izochorycznej (5.45) nie wykazują wypukłości względem $\mathbf{F} \in \mathrm{Lin}$ oraz $\operatorname{Cof}\mathbf{F} \in \mathrm{Lin}$.

W wielu przypadkach przedstawienie potencjału sprężystości względem odpowiednich funkcji $\mathbf{F}$ ułatwia analizę regularności potencjału sprężystości, zwłaszcza gdy jest on sumą trzech funkcji izotropowych zależnych od jednej zmiennej:

$$W(\mathbf{F}) = \mathcal{W}(\mathbf{F}, \operatorname{Cof}\mathbf{F}, \det\mathbf{F}) = W_1(\mathbf{F}) + W_2(\operatorname{Cof}\mathbf{F}) + W_3(\det\mathbf{F}) \tag{5.119}$$

Przy badaniu poliwypukłości funkcji $W(\mathbf{F})$, można bezpośrednio wykorzystać wyniki teoretyczne analizy funkcji wypukłych jednej zmiennej. W pracy [89] dowiedziono, że funkcje W_i są wypukłe względem odpowiednich zmiennych $\mathbf{F}$, $\operatorname{Cof}\mathbf{F}$ oraz $\det\mathbf{F} > 0$, co czyni $W(\mathbf{F})$ funkcją poliwypukłą. W praktyce teoretyczne modele hipersprężystości zazwyczaj ograniczają się do funkcji w formie 5.119. Przykłady najczęściej stosowanych funkcji wypukłych podane są w tab. 5.1, por. [54], [61], [89], [101].

Tabela 5.1. Przykładowe funkcje wypukłe jednej zmiennej, źródło: [71]

Funkcje wypukłe względem:		
$\mathbf{F}$	$\operatorname{Cof}\mathbf{F}$	$\det\mathbf{F}{=}J>0$
$\|\mathbf{F}\|^{2\alpha},\ \alpha{\geq}1$	$\|\operatorname{Cof}\mathbf{F}\|^{2\alpha},\ \alpha{\geq}1$	$J^{\alpha},\ \alpha{\geq}1 \vee \alpha{<}0$
$I_1^{\alpha}{\equiv}(\operatorname{tr}\mathbf{C})^{\alpha}{=}\|\mathbf{F}\|^{2\alpha},\ \alpha{\geq}1$	$I_2^{\alpha}{\equiv}(\operatorname{tr}\operatorname{Cof}\mathbf{C})^{\alpha}{=}\|\operatorname{Cof}\mathbf{F}\|^{2\alpha},\ \alpha{\geq}1$	$-\ln J,\ J{-}\ln J$
$\operatorname{tr}\mathbf{CM},\ \mathbf{M}{=}\mathbf{m}{\otimes}\mathbf{m},\ \|\mathbf{m}\|{=}1$	$\operatorname{tr}\operatorname{Cof}\mathbf{CM},\ \mathbf{M}{=}\mathbf{m}{\otimes}\mathbf{m},\ \|\mathbf{m}\|{=}1$	$e^{\beta J},\ \beta\in\mathbb{R}$
$\operatorname{tr}\mathbf{C}^{\alpha},\ \alpha{\geq}\frac{1}{2}$	$\operatorname{tr}\operatorname{Cof}\mathbf{C}^{\alpha},\ \alpha{\geq}\frac{1}{2}$	$(J{-}1)^{\alpha},\ \alpha{\geq}1$
$\exp(\beta I_1)$	$\exp(\beta I_2)$	$c\,J^2{-}d\,\ln J,\ c{>}0,\ d{>}0$

W szczególności izotropowe poliwypukłe potencjały hipersprężystości można stosunkowo łatwo skonstruować poprzez funkcje postaci:

$$\tilde{W}(I_1, I_2, I_3) = \tilde{W}_1(I_1) + \tilde{W}_2(I_2) + \tilde{W}_3(I_3) \tag{5.120}$$

gdzie I_i to niezmienniki tensora odkształcenia $\mathbf{C}$ lub $\mathbf{U}$. W procesie konstruowania takich funkcji bardzo pomocne jest twierdzenie Šilhavý'ego, które umożliwia budowanie funkcji poliwypukłej jako złożenia funkcji wypukłej z niemalejącymi funkcjami wypukłymi względem $\mathbf{F}$ i $\operatorname{Cof}\mathbf{F}$ oraz funkcji wypukłej względem $\det\mathbf{F} > 0$ [54]. Twierdzenie to wynika z klasycznych wyników analizy wypukłości funkcji wielu zmiennych [89] oraz definicji poliwypukłości. Wymóg, żeby funkcja była niemalejąca wynika z faktu, że złożenie niemalejącej funkcji wypukłej z funkcją wypukłą jest funkcją wypukłą [102].

Warunek przedstawienia funkcji JES przy pomocy sum trzech funkcji izotropowych zależnych od jednej zmiennej 5.119 jest wystarczający, ale nie jest konieczny, niektóre funkcje poliwypukłe nie mogą być zaprojektowane przy użyciu równania 5.36. Dokładniej mówiąc, na przykład, gdy energia zawiera składniki liniowe względem $\mathbf{F}$, aby zapisać je przez te niezmienniki, trzeba zastosować pierwiastek kwadratowy, ponieważ tensor Cauchy'ego–Greena prawego ma postać $\mathbf{C} = \mathbf{F}^T\mathbf{F}$. Pierwiastek kwadratowy nie jest jednak funkcją wypukłą. Gdy stosujemy pierwiastek do funkcji wypukłej, wcale nie musi powstać funkcja wypukła [102]. Niemniej jednak, jeśli stwierdzimy brak wypukłości względem zmiennych I_1, I_2, I_3 to z pewnością dana funkcja nie spełnia warunku poliwypukłości.

Dlatego analizowanie wypukłości względem I_1, I_2, I_3 jest użytecznym testem stabilności modelu hipersprężystego.

Dla skalarnej funkcji $f(\mathbf{F})$ wprowadza się następujące implikacje [67]:

$$\text{wypukłość} \Rightarrow \text{poliwypukłość} \Rightarrow \text{quasi-wypukłość} \Rightarrow \text{wypukłość rzędu pierwszego}\,. \tag{5.121}$$

W niniejszej pracy potencjały sprężystości badane będą dwutorowo:

1. weryfikowane będzie spełnienie warunku *rank-one convexity* (wypukłości względem kierunków rangi jeden), będącego warunkiem koniecznym poliwypukłości – sprawdzenie to wykonywane będzie dla przypadku PSO na podstawie wykresów warstwicowych JES,
2. potencjał sprężystości będzie konstruowany w zależności od niezmienników tensora deformacji, co pozwoli zapewnić poliwypukłość poprzez odpowiedni dobór funkcji składowych (por. rów. (5.120)).

5.9.5. Podsumowanie

W pracy naprężenie będzie formułowane jako gradient potencjału jednostkowej energii sprężystości, które przewidywane będzie przez modele uczenia maszynowego. Implikuje to spełnienie warunku **spójności termodynamicznej** por. rys. 5.3a. Ponadto potencjał jednostkowej energii sprężystości będzie formułowany w dziedzinie niezmienników deformacji przez co automatycznie otrzymuje się **symetryczny tensor naprężenia Cauchy'ego** por. rys. 5.3b, a takie sformułowanie zapewnia również spełnienie warunków **obiektywności** por. rys. 5.3c i **symetrii materialnej** por. rys. 5.3d.

6. Określanie relacji konstytutywnych hipersprężystości metodami regresji i sieci neuronowych

6.1. Wprowadzenie

Identyfikacja relacji konstytutywnych materiałów hipersprężystych — czyli wyznaczenie funkcji jednostkowej energii sprężystości W, której pochodne definiują odpowiedź naprężeniową materiału — jest jednym z fundamentalnych zagadnień mechaniki eksperymentalnej i obliczeniowej. Istnieje wiele przykładów klasycznych modeli hipersprężystych, w tym modele Neo-Hooke'a, Mooney'a-Rivlina, Yeoha, Demiraya i Ogdena, z których wszystkie wykorzystują z góry określone wyrażenie analityczne dla funkcji JES. Wybór postaci tych modeli jest wynikiem wieloletnich prac i badań [64], [66], ale ostatecznie to badacz podejmuje decyzję o wyborze konkretnego typu modelu, zanim dopasuje jego parametry do danych eksperymentalnych. Proces ten sprawdza się, gdy istnieje dogłębne zrozumienie zachowania rozpatrywanego materiału i modeli funkcji JES, jednak i to nie gwarantuje zadowalającego dopasowania do posiadanych danych czy poprawnej ekstrapolacji i może prowadzić zarówno do nadmiernego uproszczenia (pominięcia istotnych członów), jak i do przeparametryzowania (włączenia członów zbędnych) funkcji JES. Problem ten staje się znacznie poważniejszy w przypadku nowych materiałów. Patrząc na mnogość dostępnych modeli, wybór właściwej postaci W nie jest trywialny, a błędna decyzja na tym etapie nie może zostać skorygowana przez samą procedurę dopasowania parametrów. W niniejszej pracy skupiono się na materiałach izotropowych, dla których funkcja W zależy wyłącznie od niezmienników tensorów Cauchy'ego–Greena (I_1, I_2, I_3).

Idea zastąpienia analitycznych modeli konstytutywnych aproksymatorem uczonym z danych sięga początku lat 90. XX wieku. Pionierską pracą jest artykuł Ghaboussiego, Garretta i Wu z 1991 roku [8], w którym zaproponowano użycie sieci neuronowej typu FFNN (*feedforward neural network*) do modelowania zależności naprężenie–odkształcenie przy cyklicznym obciążeniu jednoosiowym i dwuosiowym w płaskim stanie naprężenia (PSN) betonu na podstawie danych eksperymentalnych, bez narzucania konkretnej postaci analitycznej. Późniejsze prace tego zespołu — zwłaszcza Hashash i in. [11] — pokazały, jak wytrenowaną sieć osadzić w procedurze UMAT programu Abaqus. Analogiczny cel został

osiągnięty w [103], a kod dołączony do publikacji umożliwia przekonwertowanie wytrenowanych wag i obciążeń w bibliotece PyTorch na ogólny kod FORTRAN [104]. Od tego czasu zagadnienie modelowania konstytutywnego sieciami neuronowymi rozwija się równolegle z postępem w uczeniu maszynowym: od sieci rekurencyjnych (RNN [105]) do opisu historii obciążenia, przez architektury informowane fizyką (PINN [106]), aż po współczesne podejścia oparte na operatorach neuronowych, tak więc z dużą dozą prawdopodobieństwa można przewidywać intensywny rozwój w modelowaniu konstytutywnym przy wykorzystaniu uczenia maszynowego.

6.1.1. Liniowa regresja

Jedną z metod uczenia maszynowego jest liniowa regresja, której popularność w modelowaniu konstytutywnym również wzrosła w ostatniej dekadzie, choć tak naprawdę jej nieliniowej wersji używamy od lat, np. stosując funkcję `NonlinearModelFit` w pakiecie Mathematica [107]. Obecne badania prowadzone są nad regresją rzadką (ang. *sparse regression*), która pozwala zerować współczynniki przy niestotnych członach, co umożliwia nie tylko wyznaczenie współczynników znanych modeli konstytutywnych, ale także wyłonić formę funkcji JES z założonej bazy funkcyjnej. Przykładowo, w pracy [108] Flaschel, Kumar i De Lorenzis zaproponowali algorytm EUCLID (*Efficient Unsupervised Constitutive Law Identification and Discovery*), w którym funkcja JES jest aproksymowana jako superpozycja zbioru funkcji bazowych, wykorzystując regresję rzadką L_p z progowaniem. Ponadto ich zbiór danych treningowych stanowiły dane przemieszczeniowe pochodzące z korelacji cyfrowej obrazów (DIC) oraz globalnej siły bez pomiarów naprężeń, stąd notabene nazwa *unsupervised*. Urrea-Quintero i in. rozszerzyli to podejście, łącząc algorytmy regresji rzadkiej, takie jak LASSO, LARS oraz OMP, z kryteriami informacyjnymi AIC i BIC [109].

6.1.2. Sieci neuronowe

Równolegle rozwinęły się podejścia oparte na sieciach neuronowych, w których JES jest parametryzowana warstwami sieci. Linka i in. [14] zaproponowali architekturę CANN (*Constitutive Artificial Neural Networks*), w której każdy neuron odpowiada jednej funkcji bazowej (np. Softplus, potęga) zależnej od niezmienników, co czyni architekturę interpretowalną w sensie mechaniki. Linka i Kuhl [15], [110] rozszerzyli ten pomysł, proponując nowe rodziny CANN z automatycznym odkrywaniem modelu.

Flaschel i in. [111] zaproponowali NN-EUCLID (*Neural Network-based Efficient Unsupervised Constitutive Law Identification and Discovery*), który rozszerza podejście EUCLID o sieć neuronową trenowaną w uczeniu nienadzorowanym na danych z pól przemieszczeń.

Z kolei Input Convex Neural Networks (ICNN) Amosa i in. [96] pozwalają skonstruować sieć, której wyjście jest wypukłe względem wejść dzięki odpowiedniej konstrukcji architektury (nieujemne wagi, wypukłe i niemalejące aktywacje). Klein i in. [97] oraz Linden i in. [91] zaadaptowali ICNN do mechaniki ciał hipersprężystych, stosując je na zestawie niezmienników deformacji I_1, I_2, J.

W opinii autora niniejszej pracy możliwym rozwiązaniem jest zastosowanie architektury równoległej PFNN (*Parallel Feed-forward Neural Network*), w której niezależne podsieci przetwarzają różne zestawy wejść, a ich wyjścia byłyby sumowane:

$$W = W_{\mathrm{ICNN}}(I_1, I_2, J) + \tilde{W}_1(I_1) + \tilde{W}_2(I_2) + \tilde{W}_3(J). \tag{6.1}$$

Pierwsza podsieć W_{ICNN} jest siecią ICNN wypukłą w niezmiennikach łącznie — co implikuje poliwypukłość, choć jest warunkiem silniejszym — i odpowiada za modelowanie interakcji między niezmiennikami. Pozostałe trzy podsieci $\tilde{W}_i$ są jednoargumentowe: każda musi być jedynie wypukła (i niemalejąca dla I_1, I_2), por. równanie (5.120). Ponieważ suma funkcji wypukłych jest wypukła, cały model (6.1) jest poliwypukły. Jednocześnie człony separowalne rozszerzają przestrzeń reprezentowalnych potencjałów poza klasę ograniczoną przez łączną wypukłość.

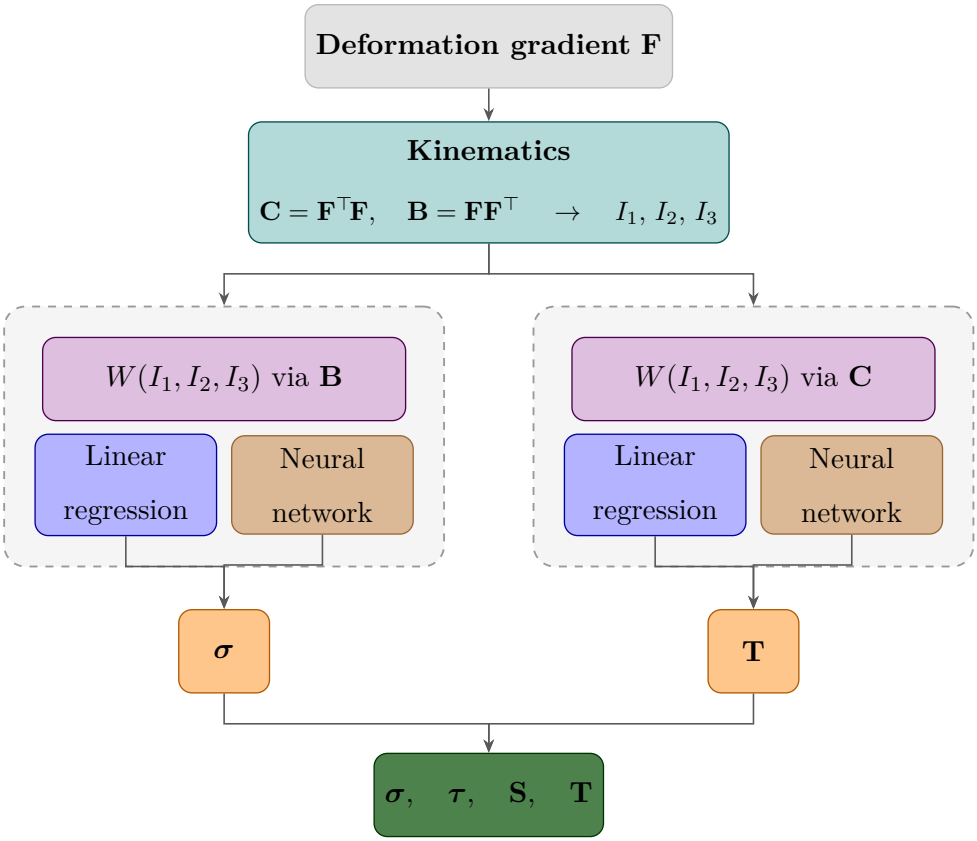
Należy podkreślić, że kod źródłowy implementacji Kleina i in. [97] oraz Lindena i in. [91] nie został udostępniony publicznie, co utrudnia niezależną weryfikację zakresu restrykcyjności przyjętej architektury. W niniejszej pracy podejście oparte na ICNN nie jest stosowane, ponieważ sieci ICNN wymagają nieliniowych funkcji aktywacji (np. Softplus), które uniemożliwiają bezpośrednie odczytanie wag jako stałych materiałowych.

Podsumowując, choć sieci neuronowe mają zdolność aproksymacji każdej funkcji ciągłej [4], [40], nie mogą być stosowane bezkrytycznie w modelowaniu konstytutywnym. Sieci neuronowe zazwyczaj składają się z wielu warstw i dlatego są określane mianem *czarnych skrzynek*. Taka sieć może bardzo dokładnie odtworzyć dane treningowe, natomiast nie dostarczy żadnego wyjaśnienia, dlaczego dany materiał zachowuje się w określony sposób, a poza zakresem danych treningowych, w przypadku ich ekstrapolacji czy przy innych stanach naprężenia jej przewidywania stają się zawodne. Modele konstytutywne muszą

nie tylko dopasowywać krzywe do danych treningowych, ale też spełniać szereg wymagań stawianych przez mechanikę ośrodków ciągłych, opisanych w sekcji 5.8, pozostając przy tym interpretowalnymi i dobrze generalizującymi.

6.1.3. Proponowane podejścia

Pierwsza część niniejszego rozdziału — obejmującego identyfikację relacji konstytutywnych metodami regresji liniowej przy użyciu niezmienników tensora deformacji — powstawała w 2022 roku, jeszcze zanim sieci PINN (omówione w rozdziale 7) stały się powszechnie stosowanym narzędziem do rozwiązywania zagadnień brzegowych mechaniki ośrodków ciągłych. Przedstawione wówczas podejście, choć konceptualnie proste, nie daje się wbudować w graf obliczeniowy sieci PINN — regresja opiera się na symbolicznym różniczkowaniu i operuje poza mechanizmem automatycznego różniczkowania. Aby umożliwić wspólną optymalizację modelu konstytutywnego z rozwiązaniem zagadnienia brzegowego, regresor zastąpiono siecią neuronową, która realizuje cały łańcuch obliczeń wewnątrz grafu PyTorcha.



Rysunek 6.1. Architektura modułu ACLE — od gradientu deformacji **F** przez kinematykę i aproksymację potencjału JES do miar naprężeń

Punkt wyjścia stanowi gradient deformacji $\mathbf{F}$, z którego wyznaczane są tensory Cauchy'ego–Greena, a następnie ich niezmienniki I_1, I_2, I_3. Niezmienniki te są zmiennymi wyrażeń algebraicznych, które to stanowią warstwę wejściową do aproksymatora funkcji JES, realizowanego regresją liniową lub siecią neuronową. Następnie naprężenia wyznaczane są poprzez różniczkowanie funkcji JES względem odpowiedniego tensora deformacji. W niniejszym rozwiązaniu funkcja jednostkowej energii sprężystości została zdefiniowana w konfiguracji materialnej przez niezmienniki prawego tensora deformacji Cauchy'ego-Greena $\mathbf{C}$, a naprężeniem wyjściowym jest drugi tensor Pioli–Kirchhoffa $\mathbf{T}$ (prawa gałąź na rys. 6.1), natomiast nic nie stoi na przeszkodzie, by zdefiniować ją przez tensor gradientu deformacji $\mathbf{F}$ czy niezmienniki lewego tensora Cauchy'ego-Greena $\mathbf{B}$. Należy jednak pamiętać, aby w ostatnim kroku wykonać odpowiednie transformacje (por. rys. 5.2) pozwalające uzyskać interesujący nas tensor naprężenia.

W swojej najbardziej podstawowej postaci sieć neuronowa wykonująca powyższe zadanie jest po prostu pojedynczą warstwą liniową bez odchylenia (ang. *bias*). Jej struktura matematyczna przypomina regresję liniową, w której każda waga sieci odpowiada współczynnikowi mnożącemu kolejne wyrażenia bazowe. Największą różnicą między tymi dwiema metodami jest sposób optymalizacji funkcji kosztu – regresja wykorzystuje metodę najmniejszych kwadratów do rozwiązania układu równań w jednym kroku, podczas gdy sieć wykorzystuje iteracyjny algorytm propagacji wstecznej. Największą korzyścią z zastosowania tego drugiego podejścia jest to, że aproksymowany model konstytutywny z pełnym różniczkowalnym grafem obliczeniowym może zostać wpięty do szerszej architektury sieci neuronowej i optymalizowany wspólnie z pozostałymi elementami w trakcie jednego przebiegu. Przykładem jego użycia może być rozwiązanie problemu brzegowego teorii sprężystości, w którym poszukiwane byłyby również parametry materiałowe.

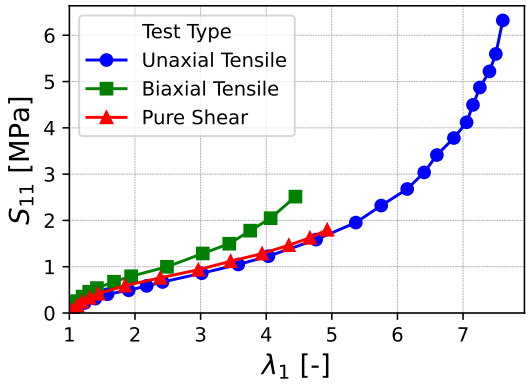
Pierwotnym planem było zastąpienie liniowej regresji taką jednowarstwową siecią neuronową, natomiast został on zastąpiony algorytmem z nieco głębszą architekturą, konstytutywnej sieci neuronowej (CANN) opracowanej przez Linkę i in.[14], [15]. Teoretycznie już płytka sieć neuronowa może przybliżać dowolną funkcję ciągłą [40], ale w pracy [112] Montufar i in. udowodnili, że potrzebuje do tego większej liczby parametrów. Decyzję poparł przykład zaprezentowany w 6.4.3, w którym zastosowano rozszerzenie funkcji JES o model Demiraya [113], w którym współczynniki stojące przy wyrażeniach

z niezmiennikami deformacji izochorycznej zostały wybrane arbitralnie, a nie w drodze optymalizacji. W CANN natomiast wspomniany współczynnik może być jedną z wag sieci.

Kod i rozwiązania przedstawione poniżej są własną, niezależną adaptacją kodu dostarczonego do pracy [15], napisanego w TensorFlow [114]. Własna implementacja została wykonana przy wykorzystaniu biblioteki PyTorch [42] i umożliwia integrację z dowolnymi architekturami napisanymi w tym samym pakiecie, w tym z rozwiązaniami prezentowanymi w rozdziale 8.

6.2. Zbiór danych Treloara

Zbiór danych z eksperymentów Treloara [33] użyty w tej pracy został opublikowany na jednym z serwisów przeznaczonych do tego celu [115]. W zbiorze można znaleźć dane pochodzące z testów jednoosiowego i dwuosiowego rozciągania oraz czystego ścinania gumy.



Rysunek 6.2. Graficzne przedstawienie danych Treloara. Wykresy naprężeń w zależności od wydłużenia dla: jednoosiowego rozciągania, dwuosiowego rozciągania, czystego ścinania

Tabela 6.1. Próbka z przetasowanego zbioru danych Treloara

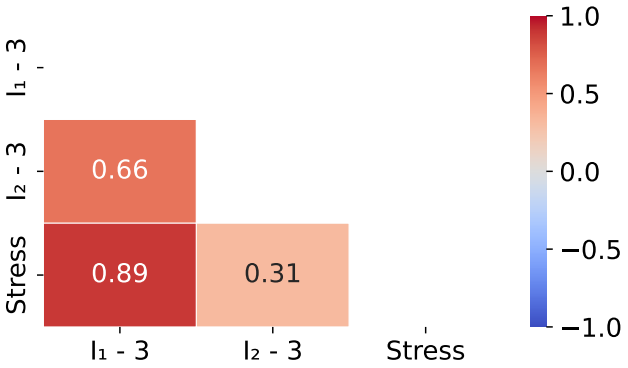
Strain [-]	Stress [MPa]	Test Type	$I_1 - 3$ [-]	$I_2 - 3$ [-]
7.054	4.119	ut	4.807	2.454
1.121	0.163	ps	0.013	0.013
2.965	0.935	ps	1.302	1.302
4.760	1.586	ut	2.677	1.574
1.306	0.331	ps	0.072	0.072

Tabela 6.1 przedstawia wydłużenie, naprężenie Piola-Kirchhoffa II rodzaju, których interpretacja została pokazana w rozdziale 5, typ testu dla każdego wiersza oraz niezmienniki deformacji, losowo przetasowanego z oryginalnego zestawu danych. Ponadto cały zbiór danych przedstawiono graficznie na rys. 6.2.

Modele konstytutywne sformułowano wyłącznie przy użyciu niezmienników deformacji I_1, I_2, które odpowiadają kolejno za rozciąganie/ściskanie oraz ścinanie materiału [116]. Niezmienniki te określono zakładając więzy nieściśliwości $I_3 = \det(\mathbf{C}) = 1$. Użycie nie-

zmienników pozwala wyrazić potencjał hipersprężysty jako funkcję niemalejącą i wypukłą w niezmiennikach, co przy $I_3 = 1$ implikuje poliwypukłość (por. rys. 5.3i). Ponadto zapewnia to symetrię tensora naprężeń (por. rys. 5.3b), obiektywność (por. rys. 5.3c), symetrię materialną (por. rys. 5.3d) oraz zgodność termodynamiczną (por. rys. 5.3a)[91].

Na rysunku 6.3 przedstawiono macierz korelacji Pearsona dla całego zbioru danych. Wynika z niej, że naprężenie silnie koreluje z pierwszym niezmiennikiem ($r = 0.89$) i umiarkowanie słabo z drugim ($r = 0.31$), natomiast korelacja między niezmiennikami ($r = 0.66$) oznacza, że I_2 wnosi informację częściowo niezależną od I_1.



Rysunek 6.3. Macierz korelacji Pearsona

Dane pierwotnie zostały opublikowane w formie wykresów. Po odczytaniu punktów z wykresów można przeprowadzić interpolację kwadratową dla każdej deformacji, co generuje równomiernie rozmieszczone punkty na całym zakresie odkształceń. To podejście wydaje się praktyczne, gdy pracujemy z danymi dostępnymi wyłącznie w formie graficznej. Jeżeli dodatkowo ujednolici się liczbę punktów dla każdego typu testu, żaden przypadek deformacji nie dominuje nad pozostałymi, a każdy test ma równy wpływ na identyfikację parametrów. Oryginalnie odczytane punkty z wykresów stanowią zbiór testowy i pozostają niezależne od przyjętej liczby punktów interpolacji, a ocena modelu odbywa się na rzeczywistych danych pomiarowych, a nie na punktach sztucznie wygenerowanych przez interpolację. Kwestia doboru liczby punktów interpolacji zostanie omówiona w dalszej części pracy.

6.3. Wybór rozwinięcia jednostkowej energii sprężystości

W przypadku materiałów hipersprężystych $W = W(\mathbf{C})$ funkcję jednostkowej energii sprężystości, bazującą na niezmiennikach, wygodnie jest rozwijać w ich potęgi. W pracy

przyjęto ogólną postać:

$$\hat{W}(I_1, I_2, J) = \sum_{k+l+m=1}^{N} C_{k,l,m} \Big(I_1 - 3\Big)^k \Big(I_2 - 3\Big)^l \Big(J - 1\Big)^m, \tag{6.2}$$

Przesunięcia niezmienników tj. $(I_1 - 3)$, $(I_2 - 3)$, $(J - 1)$ zapewniają $\tilde{W} = 0$ w stanie odniesienia ($\mathbf{F} = \mathbf{I}$). W dalszych etapach pracy funkcję (6.2) rozwinięto o człony Demiraya [113]:

$$\tilde{W}(I_1, I_2) = \sum_{k=1}^{K} \beta_{k,0} \left[\exp(\alpha_{k,0} (I_1 - 3)) - 1\right] + \sum_{k=1}^{K} \tilde{\beta}_{0,k} \left[\exp(\alpha_{0,k} (I_2 - 3)) - 1\right], \tag{6.3}$$

w taki sposób, że całkowita funkcja JES przyjmowała postać $\hat{W} + \tilde{W}$.

Przez odpowiedni dobór współczynników $C_{k,l,m}$ i zamianę zmiennych niezmienników deformacji na niezmienniki deformacji izochorycznej [78], [83], [117] rozwinięcie (6.2) pozwala uzyskać pełną rozdzielność na część izochoryczną $\bar{W}_D(\bar{I}_1, \bar{I}_2)$ i objętościową $\bar{W}_I(J)$ oraz zredukować się do wielu klasycznych form opisujących zarówno materiały nieściśliwe ($J = 1$) jak i mało ściśliwe. Jednym z przykładów jest zmodyfikowana wielomianowa propozycja Rivlina [77], [118]:

$$\bar{W}(\bar{I}_1, \bar{I}_2, J) \;=\; \sum_{k+l=1}^{N} C_{k,l,0} \, (\bar{I}_1 - 3)^k \, (\bar{I}_2 - 3)^l \;+\; \sum_{k=1}^{N} C_{0,0,2k} \, (J - 1)^{2k}. \tag{6.4}$$

W przypadku rozpatrywania materiału w programie ABAQUS [80] należy przyjąć:

$$C_{k,l,0} = C_{kl}, \quad C_{0,0,2k} = \frac{1}{D_k}, \tag{6.5}$$

gdzie parametry D_k określają ściśliwość materiału (dla $D_k \to 0$ otrzymujemy model materiału ściśliwego). W przypadku materiału nieściśliwego należy w danych do programu ABAQUS podstawić bliskie zeru parametry D_k. Ponieważ w niniejszej pracy zbiór danych doświadczalnych Treloara został stworzony w oparciu o związki nieściśliwości, w dalszych częściach będzie stosowana skrócona notacja C_{kl} zamiast $C_{k,l,0}$. Należy zaznaczyć, że szczególnymi przypadkami modelu wg propozycji (6.4) są między innymi następujące modele materiałów mało ściśliwych / nieściśliwych:

Dodatkowym, istotnym argumentem przemawiającym za przyjęciem wielomianowego rozwinięcia (6.2) jest możliwość łatwego skonstruowania poliwypukłego potencjału ES (zob.

Tabela 6.2. Wybrane modele ES jako szczególne przypadki ogólnego rozwinięcia (6.2)

L.p.	Model	Postać funkcji ES nieściśliwych [mało ściśliwych]
1	neo-Hooke [77]	$\bar{W} = C_{1,0,0}(\bar{I}_1 - 3) + \left[(C_{0,0,2}(J-1)^2\right]$
2	Yeoh [16]	$\bar{W} = \sum_{k=1}^{3} C_{k,0,0}\,(\bar{I}_1 - 3)^k + \left[\sum_{k=1}^{3} C_{0,0,2k}\,(J-1)^{2k}\right]$
3	Mooney–Rivlin [77]	$\bar{W} = C_{1,0,0}\,(\bar{I}_1 - 3) + C_{0,1,0}\,(\bar{I}_2 - 3) + C_{0,0,2}\,(J-1)^2$
4	Biderman [119]	$\bar{W} = \sum_{k=1}^{3} C_{k,0,0}\,(\bar{I}_1 - 3)^k + C_{0,1,0}\,(\bar{I}_2 - 3)$

podrozdział 5.9.4, s. 95, szczególnie równanie (5.120)). Nasze rozwinięcie (6.2) funkcji ES przyjęłoby wówczas:

$$\hat{W}(I_1, I_2, J) = \sum_{k=1}^{N} C_{k,0,0}\,(I_1 - 3)^k + \sum_{l=1}^{N} C_{0,l,0}\,(I_2 - 3)^l + \sum_{m=1}^{N} C_{0,0,m}\,(J-1)^m \tag{6.6}$$

Warunek poliwypukłości przekłada się bezpośrednio na ograniczenia dla $C_{k,l,m}$:

(i)	$C_{k,0,0} \geq 0$	$k \geq 1$
(ii)	$C_{0,l,0} \geq 0$	$l \geq 1$
(iii)	$C_{0,0,m} \geq 0$	$m \geq 1^*$

(6.7)

Warunki (6.7)(i)–(ii) gwarantują, że wielomiany $\hat{W}_1(I_1), \hat{W}_2(I_2)$ są wypukłe i niemalejące dla $I_1, I_2 \geq 3$ [54], [89], natomiast wymóg (6.7)(iii) zapewnia wypukłość względem $\det \mathbf{F} > 0$ [54]. *Wyraz liniowy $(J-1)$ decyduje wyłącznie o ciśnieniu w konfiguracji początkowej i jeśli żąda się stanu beznaprężeniowego, przyjmuje się $C_{0,0,1} = 0$ [66], [120].

6.4. Implementacja modelu regresji z wykorzystaniem niezmienników deformacji

W niniejszym rozdziale omówione zostanie, w jaki sposób niezmienniki deformacji mogą być przekształcane aby formułować, trenować i zapisywać wagi modelu regresji liniowej. W celu efektywnego i spójnego przekształcania danych w procesie modelowania wykorzystano moduły przekształcenia danych (`transformers`). Są to estymatory udostępnione w pakiecie `scikit-learn` [25] obsługujące metody `transform` i/lub `fit_transform`. Aby zbudować estymator złożony, moduły te łączy się z innymi modułami przekształceń lub

predyktorami (takimi jak klasyfikatory lub regresory). Do kreowania estymatorów złożonych zastosowano potok danych (`pipeline`). Potok danych wymaga, aby wszystkie etapy poza ostatnim realizowały operacje przekształceń danych. Ostatni krok w potoku może pełnić rolę modułu przekształceń, klasyfikatora lub — tak jak w omawianym przypadku — regresora. Dzięki takiemu podejściu, polegającemu na łańcuchowym dopasowywaniu kolejnych operacji przekształceń, możliwe jest odwzorowanie dowolnie wyrażonych zależności naprężenie–wydłużenie.

Rysunek 6.4. Schemat blokowy omawianego potoku danych

Do potoku danych wprowadzono dwa niestandardowe moduły przekształceń danych: `TensorToInvariants Transformer` i `DesignMatrixTransformer`, a następnie przeprowadzono liniową regresję za pomocą `LinearRegression` dostępną w pakiecie `scikit-learn`:

- `TensorToInvariantsTransformer`: moduł przekształceń danych, który pobiera tensory gradientu deformacji $\mathbf{F}$, przekształca je w prawe tensory deformacji Cauchy'ego-Greena $\mathbf{C}$, a następnie wyznacza niezmienniki deformacji I_1, I_2 i J,
- `DesignMatrixTransformer`: moduł przekształceń danych będący głównym elementem tego algorytmu, którego celem jest utworzenie macierzy wielomianowych cech naprężeniowych i obliczanie symbolicznych pochodnych funkcji jednostkowej energii sprężystości względem niezmienników deformacji I_1, I_2 i J. Wykorzystuje `FeatureGenerator` do tworzenia cech wielomianowych i eksponencjalnych o zadanym stopniu omówionych w 6.3, a następnie wykorzystuje bibliotekę `sympy` [121] do obliczania pochodnych symbolicznych takich cech. Pochodne te są przekształcane w funkcje numeryczne, a następnie wykorzystywane do budowy macierzy cech naprężeniowych, która jest używana w regresji liniowej,
- `LinearRegression`: regresja liniowa próbuje znaleźć zestaw współczynników (parametrów modelu), które najlepiej opisują zależność liniową pomiędzy cechami naprężeniowymi a zmienną docelową - tensorem naprężenia Pioli-Kirchhoffa I rodzaju. Parametry te są szacowane metodą najmniejszych kwadratów, minimalizując sumę kwadratów różnic pomiędzy przewidywaniami modelu a rzeczywistymi wartościami zmiennej docelowej.

Kod modułów przekształceń danych dostępny jest w załączniku 3.

Konstrukcja macierzy cech naprężeniowych przy użyciu transformera `DesignMatrixTransformer`

W zadaniach przedstawionych w niniejszym rozdziale kluczowym elementem jest tworzenie macierzy cech naprężeniowych (*design matrix*), która pozwoli na zastosowanie odpowiednich metod estymacji parametrów w modelu materiałowym. Macierz cech naprężeniowych jest wejściem do wszystkich modeli liniowej regresji opisanych w tym rozdziale. Klasa ta korzysta z wielomianowych rozwinięć funkcji jednostkowej energii sprężystości w dziedzinie niezmienników deformacji oraz oblicza pochodne cząstkowe tej funkcji względem nich.

Konstruktor tej klasy przyjmuje dwa kluczowe elementy:

- **degree** – stopień wyrażeń algebraicznych, za pomocą których generowana będzie funkcja jednostkowej energii sprężystości,
- **model_fit** – wariant obliczeń macierzy cech naprężeniowych. Dopuszczalnymi wartościami są:
  - *simplified* – uproszczony model dopasowuje się do pierwszej składowej tensora naprężeń S_{11},
  - *full* – w tym przypadku model dopasowuje się do wszystkich składowych tensora naprężenia $\mathbf{S}$.

Metoda `fit` inicjuje konfigurację generatora cech wielomianowych na podstawie danych wejściowych, wykorzystując niezmienniki I_1, I_2, J przekazane w polu `X['inv']`.

Głównymi krokami tej metody są:

1. tworzenie funkcji bazowych φ_α, gdzie $\alpha = 1, \ldots, N_{\text{cech}}$, w zależności od wybranego rozwinięcia wg podsekcji 6.3. Przykładowy wydruk z klasy generującej funkcje bazowe (`FeatureGenerator`) został zamieszczony w załączniku 4.
2. dla każdego wyrażenia φ_α obliczenie pochodnych cząstkowych względem kolejnych niezmienników:
$$\frac{\partial \varphi_\alpha}{\partial I_1}, \quad \frac{\partial \varphi_\alpha}{\partial I_2}, \quad \frac{\partial \varphi_\alpha}{\partial J} \tag{6.8}$$
i przekształcanie ich w funkcje numeryczne.

Dzięki takiemu podejściu podczas dalszego przetwarzania, w metodzie `transform`, możliwe jest szybkie obliczanie wartości funkcji φ_α i ich pochodnych dla dowolnego zestawu wartości (I_1, I_2, J). Głównymi krokami metody `transform` są:

1. na podstawie wcześniej zdefiniowanych funkcji numerycznych, dla każdego kroku obciążenia n obliczane są wartości pochodnych:

$$\left.\frac{\partial \varphi_\alpha}{\partial \bar{I}_1}\right|^{(n)}, \quad \left.\frac{\partial \varphi_\alpha}{\partial \bar{I}_2}\right|^{(n)}, \quad \left.\frac{\partial \varphi_\alpha}{\partial J}\right|^{(n)}. \tag{6.9}$$

2. konstrukcja macierzy cech naprężeniowych — na podstawie obliczonych wartości pochodnych metoda `transform` wyznacza tensor cech w mierze Pioli-Kirchhoffa II rodzaju (por. zał. 2):

$$\mathbf{T}_\alpha^{(n)} = 2\left[\left(\left.\frac{\partial \varphi_\alpha}{\partial \bar{I}_1}\right|^{(n)} + \bar{I}_1^{(n)} \left.\frac{\partial \varphi_\alpha}{\partial \bar{I}_2}\right|^{(n)}\right)\mathbf{I} - \left.\frac{\partial \varphi_\alpha}{\partial \bar{I}_2}\right|^{(n)} \mathbf{C}^{(n)} + \frac{J^{(n)}}{2}\left.\frac{\partial \varphi_\alpha}{\partial J}\right|^{(n)} \left(\mathbf{C}^{(n)}\right)^{-1}\right], \tag{6.10}$$

gdzie $\mathbf{I}$ jest macierzą jednostkową 3×3, a $\left(\mathbf{C}^{(n)}\right)^{-1}$ oznacza macierz odwrotną do $\mathbf{C}^{(n)}$. Dane doświadczalne wyrażone są w mierze Pioli-Kirchhoffa I rodzaju, tensor ten jest następnie odwzorowywany na odpowiednią miarę:

$$\mathbf{S}_\alpha^{(n)} = \mathbf{F}^{(n)}\,\mathbf{T}_\alpha^{(n)}. \tag{6.11}$$

Wiersze wynikowej macierzy cech naprężeniowych odpowiadają kolejnym krokom obciążenia n, a kolumny — tensorom $\mathbf{T}_\alpha^{(n)}$ dla kolejnych funkcji bazowych φ_α, które w implementacji przyjmują spłaszczoną reprezentację macierzową.

Uwaga: wzór (6.10) został przedstawiony dla ogólnego przypadku materiału ściśliwego i należy go dopasować do przyjęto modelu materiałowego zgodnie z zał. 2.

Użycie transformera zwracającego naprężeniową macierz cech pozwala nam interpretować współczynniki stojące przy cechach jako parametry (wagi) funkcji jednostkowej energii sprężystości. Ponadto umożliwia zdefiniowanie funkcji jednostkowej energii sprężystości za pomocą dowolnych funkcji w dziedzinie niezmienników deformacji. W celu rozjaśnienia działania `DesignMatrixTransformer` w załączniku 3 zamieszczono kod źródłowy tej klasy, a w załączniku 5 przeprowadzono studium przypadku dla próby jednoosiowej i założenia nieściśliwości materiału opisanego na bazie wielomianów co najwyżej stopnia drugiego.

6.4.1. Liniowa regresja z użyciem macierzy cech naprężeniowych dla modeli zaimplementowanych w systemie ABAQUS

W niniejszej sekcji sprawdzono działanie przygotowanego potoku obliczeniowego (opisanego w sekcji 6.4). W tym celu wyznaczono współczynniki wielomianów funkcji JES dla modeli dostępnych w pakiecie ABAQUS [80] (zob. tab. 6.2):

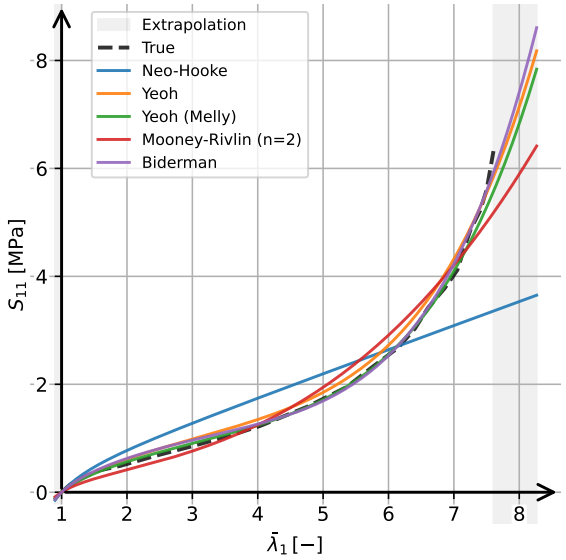
- Neo–Hooke,
- Yeoh,
- uogólnionego Mooneya–Rivlina ($n = 2$),
- Bidermana.

Parametry identyfikowano jednocześnie na danych z jednoosiowego rozciągania, dwuosiowego rozciągania oraz czystego ścinania. W niniejszym przykładzie przyjęto $n_\mathrm{p} = 40$ punktów interpolacji dla każdego przypadku deformacji. Wyznaczone wartości zestawiono w tabeli 6.3, w której poszczególnym konfiguracjom modeli przypisano oznaczenia wersji zgodnie z konwencją opisaną w rozdziale 3.4.6.

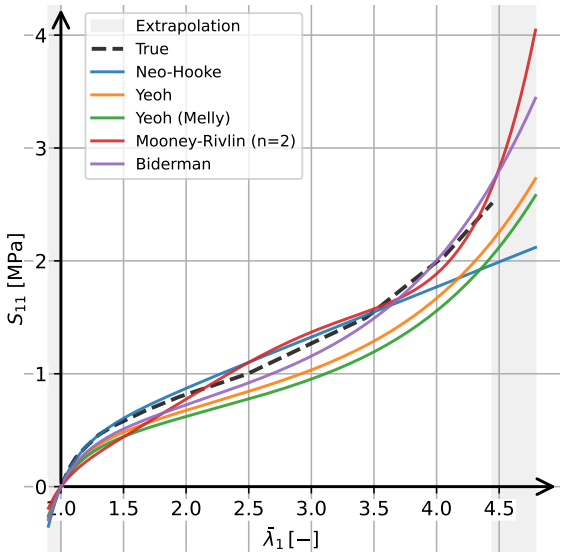
Tabela 6.3. Parametry modeli hipersprężystości wyznaczone metodą regresji liniowej na danych Treloara [33]. Dla porównania podano parametry modelu Yeoha wg Melly'ego [122]

Wersja	**Model**	**Parametry [MPa]**
v1.0.0.1	Neo–Hooke	$C_{10} = 0.2212$
v1.0.0.2	Yeoh	$C_{10} = 0.1791$
		$C_{20} = -1.000 \times 10^{-3}$
		$C_{30} = 3.472 \times 10^{-5}$
v1.0.0.2-melly	Yeoh (Melly [122])	$C_{10} = 0.1649$
		$C_{20} = -9.556 \times 10^{-4}$
		$C_{30} = 3.376 \times 10^{-5}$
v1.0.0.3	Mooney–Rivlin ($n = 2$)	$C_{10} = 0.0972$
		$C_{01} = 0.0282$
		$C_{20} = 2.411 \times 10^{-3}$
		$C_{11} = -1.316 \times 10^{-3}$
		$C_{02} = 6.050 \times 10^{-5}$
v1.0.0.4	Biderman	$C_{10} = 0.1842$
		$C_{01} = 3.706 \times 10^{-3}$
		$C_{20} = -1.825 \times 10^{-3}$
		$C_{30} = 4.467 \times 10^{-5}$

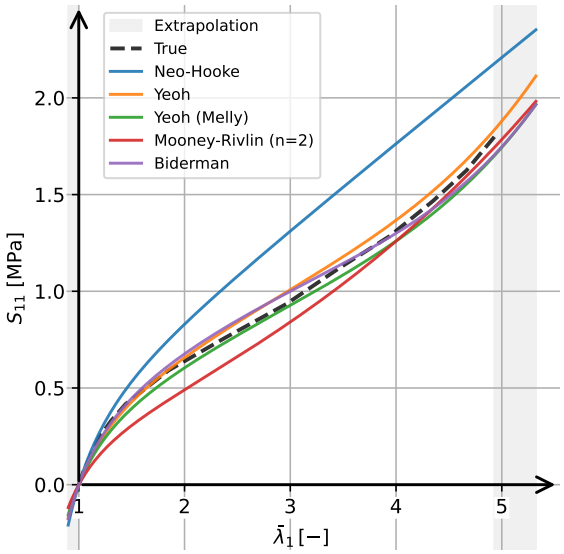
Zaproponowane rozwiązanie zostało zweryfikowane na parametrach modelu Yeoha wyznaczonego na podstawie testu jednoosiowego rozciągania przez Melly'ego w pracy [122]. Współczynniki te nadpisano w wytrenowanym potoku obliczeniowym dla modelu Yeoha i porównano krzywe naprężenie–wydłużenie z [122] z tymi uzyskanymi przez algorytm.



(a) Jednoosiowe rozciąganie



(b) Dwuosiowe rozciąganie



(c) Czyste ścinanie

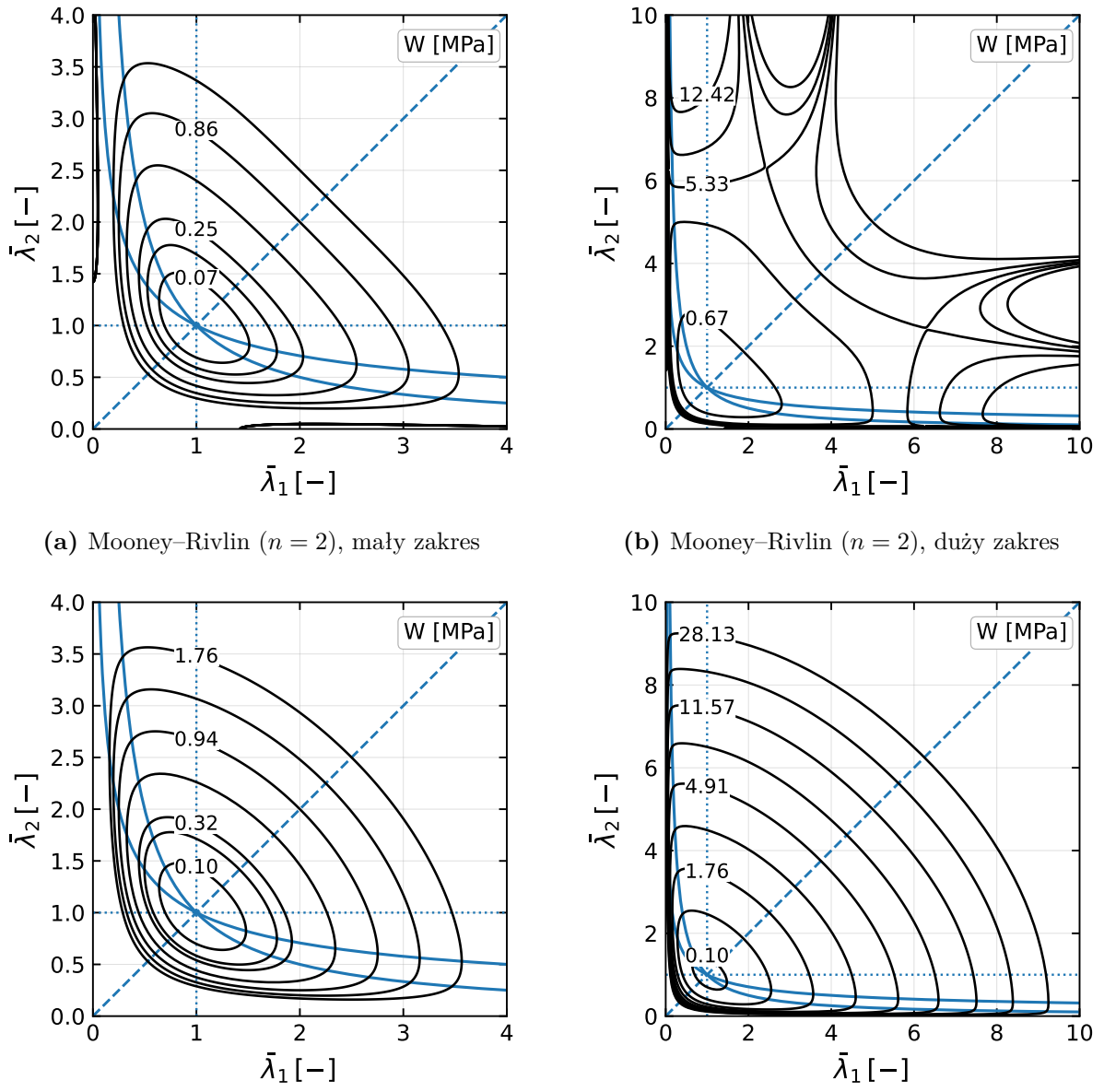
Rysunek 6.5. Wykresy S_{11} względem wydłużenia λ_1: porównanie modeli hipersprężystości wyznaczonych zaproponowanym algorytmem regresji liniowej na tle danych eksperymentalnych

Na rysunku 6.5 można zauważyć, że model Neo-Hooke'a nie jest w stanie odtworzyć s-kształtnej krzywej naprężenie-wydłużenie. Potwierdza to niska wartość współczynnika determinacji $R^2 = 0.794$. Wszystkie inne modele osiągają znacznie lepsze dopasowanie do danych. Wyniki znajdują się w tabeli 6.4. Najlepszy wynik uzyskał model Bidermana z $R^2 = 0.997$ oraz MSE $= 0.008$ [MPa^2].

Tabela 6.4. Metryki testowe modeli hipersprężystych wyznaczonych metodą regresji liniowej – uszeregowane od najlepszego do najgorszego

Wersja	**Model**	**R^2(test)**	**MSE(test) [MPa]**
v1.0.0.4	Biderman	0.997	0.008
v1.0.0.2	Yeoh	0.991	0.023
v1.0.0.2-melly	Yeoh (Melly)	0.987	0.034
v1.0.0.3	Mooney–Rivlin ($n = 2$)	0.978	0.055
v1.0.0.1	Neo–Hooke	0.794	0.514

Wykres warstwicowy gęstości energii sprężystości $W(\bar{\lambda}_1, \bar{\lambda}_2)$ w przestrzeni głównych wydłużeń pozwala na wizualną weryfikację wypukłości funkcji energii w przypadku PSO. Prawidłowy kształt konturów, zamknięte, wypukłe poziomice monotonicznie rosnące wraz z oddalaniem się od stanu nieodkształconego $(\bar{\lambda}_1, \bar{\lambda}_2) = (1, 1)$, jest warunkiem koniecznym poliwykłości. Na rysunkach 6.6a– 6.6d przedstawiono wykresy warstwicowe dla modelu Mooney-Rivlina ($n = 2$, v1.0.0.3) oraz modelu Bidermana (v1.0.0.4) w dwóch zakresach odkształceń.



(a) Mooney–Rivlin ($n = 2$), mały zakres

(b) Mooney–Rivlin ($n = 2$), duży zakres

(c) Biderman, mały zakres

(d) Biderman, duży zakres

Rysunek 6.6. Wykresy warstwicowe funkcji jednostkowej energii sprężystości $W(\bar{\lambda}_1, \bar{\lambda}_2)$ w przypadku deformacji izochorycznej dla modelu Mooney–Rivlina ($n = 2$, v1.0.0.3) oraz modelu Bidermana (v1.0.0.4). Niebieskimi liniami zaznaczono krzywe odpowiadające podstawowym testom doświadczalnym: jednoosiowemu rozciąganiu, dwuosiowemu rozciąganiu oraz czystemu ścinaniu, a linią przerywaną — przekątną $\bar{\lambda}_1 = \bar{\lambda}_2$.

W przypadku modelu Mooney-Rivlina ($n = 2$) występują wyraźne naruszenia wypukłości. Kontury mają kształty wklęsłe i pojawiają się samoprzecięcia linii poziomicowych, co szczególnie można zaobserwować przy szerszym zakresie wydłużeń (rys. 6.6b). Zachowanie to wynika z obecności członu $(\bar{I}_1 - 3)(\bar{I}_2 - 3)$ w funkcji energii. Iloczyn dwóch wyrażeń, z których każde osobno jest wypukłe, nie musi być wypukły w całej dziedzinie. To wskazuje

na naruszenie warunku poliwypukłości i dyskwalifikuje model z zastosowań wymagających stabilności numerycznej przy dużych deformacjach. Pozostałe rozważane modele miały poprawny kształt konturów, a jako przykład poprawnego zachowania przedstawiono model Bidermana (rys. 6.6c–6.6d).

6.4.2. Regularyzacja i identyfikacja istotnych składników JES

Dotychczas stopień wielomianu funkcji JES był dobierany *a priori* na podstawie przyjętej postaci modelu materiałowego. Takie podejście wymaga od nas wcześniejszego założenia, które człony funkcji JES są rzeczywiście fizycznie istotne. Inną możliwością jest przyjęcie wielomianu o dość dużym stopniu n i postawić zadanie wyzerowania nieistotnych współczynników. Innymi słowy zamiast podejmować decyzję, jakiego stopnia wielomian wybrać, możemy odpowiedzieć na pytanie *„które człony wielomianu o stopniu n są rzeczywiście potrzebne?”*

W tym celu należy dodać do algorytmu zwykłej regresji liniowej człon regularyzujący, tzn. penalizujący algorytm za przyjęcie nadmiernej liczby parametrów. W uczeniu maszynowym regularyzację stosuje się przede wszystkim w celu zapobiegania przeuczeniu (ang. *overfitting*). Model o zbyt dużej liczbie parametrów dopasowuje się zbyt mocno do szumu w danych treningowych i traci zdolność generalizacji. Dwoma powszechnie używanymi metodami regularyzacji są regresja grzbietowa (ang. *Ridge*, kara L_2):

$$L_2(\underline{w}) = \left\| \underline{y} - \underline{\underline{X}}\,\underline{w} \right\|_2^2 + \alpha \left\| \underline{w} \right\|_2^2 , \tag{6.12}$$

oraz regresja Lasso (ang. *Least Absolute Shrinkage and Selection Operator*, kara L_1):

$$L_1(\underline{w}) = \frac{1}{2n_\mathrm{p}} \left\| \underline{y} - \underline{\underline{X}}\,\underline{w} \right\|_2^2 + \alpha \left\| \underline{w} \right\|_1 , \tag{6.13}$$

gdzie $\alpha > 0$ jest parametrem regularyzacji kontrolującym siłę kary. Wzory (6.12)–(6.13) zostały przytoczone zgodnie z konwencją przyjętą w dokumentacji biblioteki `scikit-learn`. Czynnik $\frac{1}{2n_\mathrm{p}}$ przy członie kwadratowym w równaniu (6.13) upraszcza zapis pochodnej używanej wewnętrznie przez Lasso. Ponadto uniezależnia optymalną wartość α od liczby punktów pomiarowych n_p. Porównując wartości α w dwóch metodach, należy mieć na uwadze, że regresja grzbietowa tego czynnika nie zawiera.

Obie metody zmniejszają wartości mało istotnych współczynników, ale tylko kara L_1 przy wystarczająco dużym α zeruje mało istotne współczynniki. Ta cecha decyduje przy wyborze regularyzacji Lasso w niniejszej pracy–algorytm sam wskazuje, które człony przy $(\bar{I}_1 - 3)^i(\bar{I}_2 - 3)^j$ są niezbędne do zamodelowania relacji konstytutywnej, a które mogą zostać pominięte.

Kara L_1 nakładana jest jednakowo na wszystkie wagi C_{ijk}, więc jej relatywny wpływ zależy od skali kolumn macierzy $\underline{X}$. Zauważmy, że norma w macierzy naprężeniowej cech kolumny $(\bar{I}_1 - 3)^9$ jest kilka rzędów wielkości większa niż norma kolumny $(\bar{I}_1 - 3)^1$ (jest proporcjonalna do i-tej potęgi wartości niezmiennika, więc rośnie wykładniczo wraz ze zwiększaniem stopnia wielomianu). W celu zobrazowania problemu przyjrzyjmy się wzorowi (6.13) i załóżmy, że norma kolumny $(\bar{I}_1 - 3)^9$ jest 10^9 razy większa od normy kolumny $(\bar{I}_1 - 3)^1$. W takim przypadku 10^9 mniejsza waga przy wyższej potędze 9 daje dokładnie tak samo duży wkład w dopasowanie jak waga przy potędze 1. Lasso dąży do utrzymania jakości dopasowania przy niskich wagach współczynników, które dają niski koszt regularyzacji, co przy tak postawionym zadaniu eliminowałoby wagi przy niskich potęgach wyrażeń z niezmiennikami, które są fizycznie najistotniejsze.

Rozwiązaniem jest standaryzacja kolumn macierzy naprężeniowej cech przed wykonaniem optymalizacji parametrów algorytmem Lasso. Zastosowano `StandardScaler(with_mean=False)`, który dzieli każdą kolumnę przez jej odchylenie standardowe wyznaczone na zbiorze treningowym:

$$\tilde{X}_{ik} = \frac{X_{ik}}{\hat{\sigma}_k}, \qquad \hat{\sigma}_k = \sqrt{\frac{1}{n_\mathrm{p}} \sum_{i=1}^{n_\mathrm{p}} X_{ik}^2}. \tag{6.14}$$

Parametr `with_mean=False` jest konieczny, gdyż odjęcie średniej naruszyłoby warunek $W = 0$ oraz $\boldsymbol{S} = \mathbf{0}$ w stanie nieodkształconym. Po treningu współczynniki do pierwotnej skali odtwarza się przez $C_{ijk} = \tilde{C}_{ijk}/\hat{\sigma}_{ijk}$. Wektora naprężeń $\underline{y}$ nie skaluje się, gdyż uzyskiwane wyniki są zadowalające bez tego zabiegu, a rezygnacja z niego upraszcza odtworzenie C_{ijk}.

Regresja Lasso ze standaryzacją została wstawiona do potoku obliczeniowego i została przetestowana na następującym wielomianie:

$$\hat{W}(\bar{I}_1, \bar{I}_2) \;=\; \sum_{k=1}^{9} C_{k,0,0}\,(\bar{I}_1 - 3)^k \;+\; \sum_{l=1}^{9} C_{0,l,0}\,(\bar{I}_2 - 3)^l \tag{6.15}$$

Przeprowadzono trzy eksperymenty różniące się sposobem doboru parametru regularyzacji α:

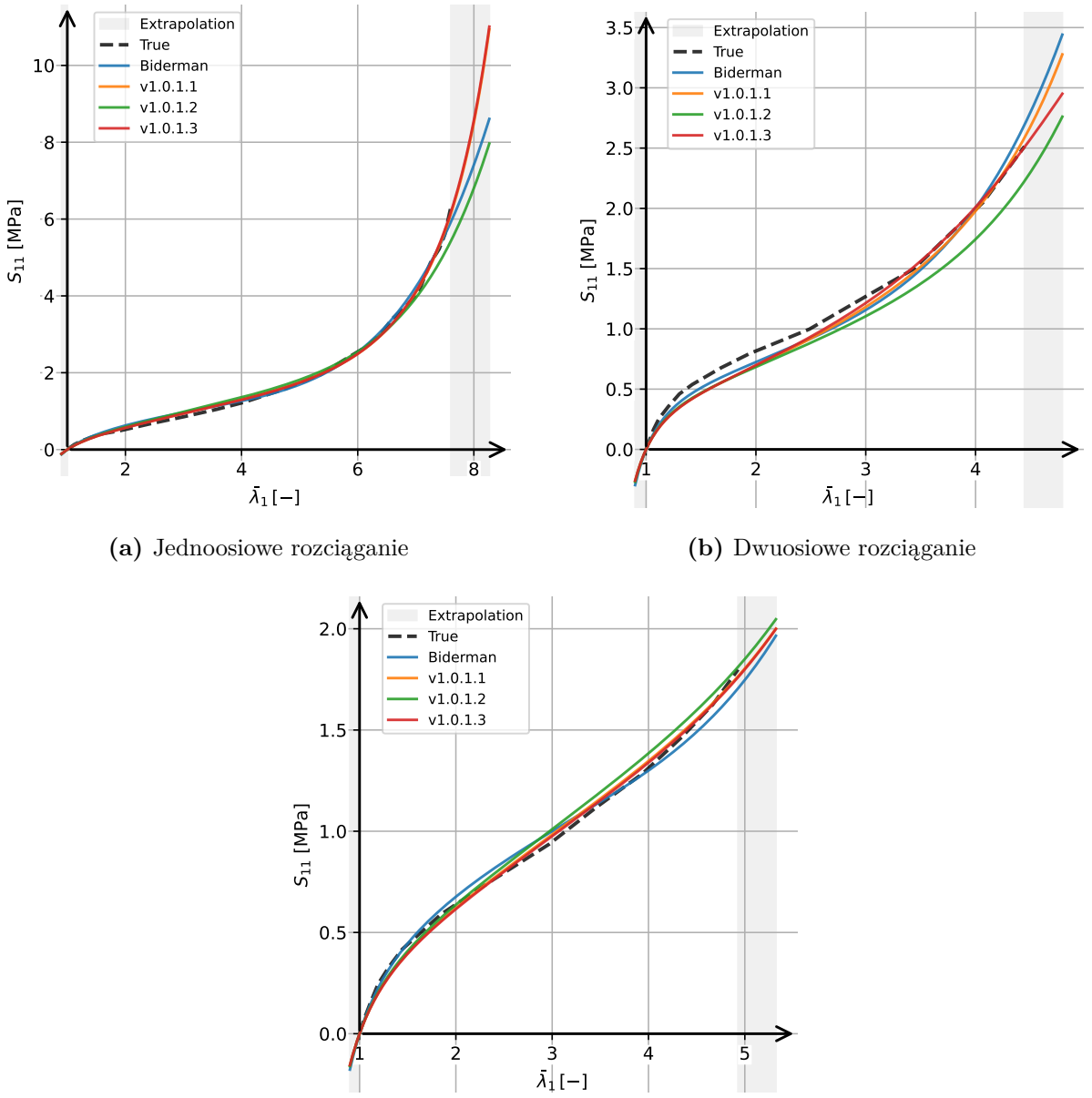
- v1.0.1.1 — $\alpha = 10^{-2}$, dobrany ręcznie,
- v1.0.1.2 — $\alpha = 10^{-1}$, dobrany ręcznie,
- v1.0.1.3 — α dobrany automatycznie metodą pięciokrotnej walidacji krzyżowej (`LassoCV`, siatka 100 wartości). Wybrana wartość: $\alpha = 3.41 \times 10^{-3}$.

Otrzymano następujące rezultaty:

Tabela 6.5. Parametry modeli hipersprężystości wyznaczone metodą regresji Lasso na danych Treloara [33]

Wersja	**Model**	**Parametry [MPa]**
v1.0.1.1	Lasso ($\alpha = 10^{-2}$)	$C_{10} = 0.1621$
		$C_{01} = 3.437 \times 10^{-3}$
		$C_{30} = 3.487 \times 10^{-7}$
		$C_{40} = 2.975 \times 10^{-7}$
		$C_{90} = 5.212 \times 10^{-17}$ †
v1.0.1.2	Lasso ($\alpha = 10^{-1}$)	$C_{10} = 0.1686$
		$C_{01} = 1.229 \times 10^{-3}$
		$C_{30} = 1.847 \times 10^{-6}$
		$C_{40} = 2.574 \times 10^{-7}$
v1.0.1.3	LassoCV ($\alpha = 3.41 \times 10^{-3}$)	$C_{10} = 0.1598$
		$C_{01} = 4.296 \times 10^{-3}$
		$C_{03} = -2.901 \times 10^{-9}$ †
		$C_{30} = 9.969 \times 10^{-7}$
		$C_{40} = 2.965 \times 10^{-7}$
		$C_{90} = 5.194 \times 10^{-17}$ †

Symbolem † oznaczono współczynniki pojawiające się względem wersji modelu v1.0.1.2 z największą siłą regularyzacji. Poniżej przedstawiono krzywe naprężenie–wydłużenie, które zestawiono z danymi eksperymentalnymi i najlepszym modelem liniowej regresji z funkcją JES Bidermana.



(a) Jednoosiowe rozciąganie

(b) Dwuosiowe rozciąganie

(c) Czyste ścinanie

Rysunek 6.7. Wykresy S_{11} względem wydłużenia λ_1: porównanie modeli uzyskanych regresją Lasso z danymi eksperymentalnymi Treloara [33] oraz modelem Bidermana wyznaczonym regresją liniową (v1.0.0.4)

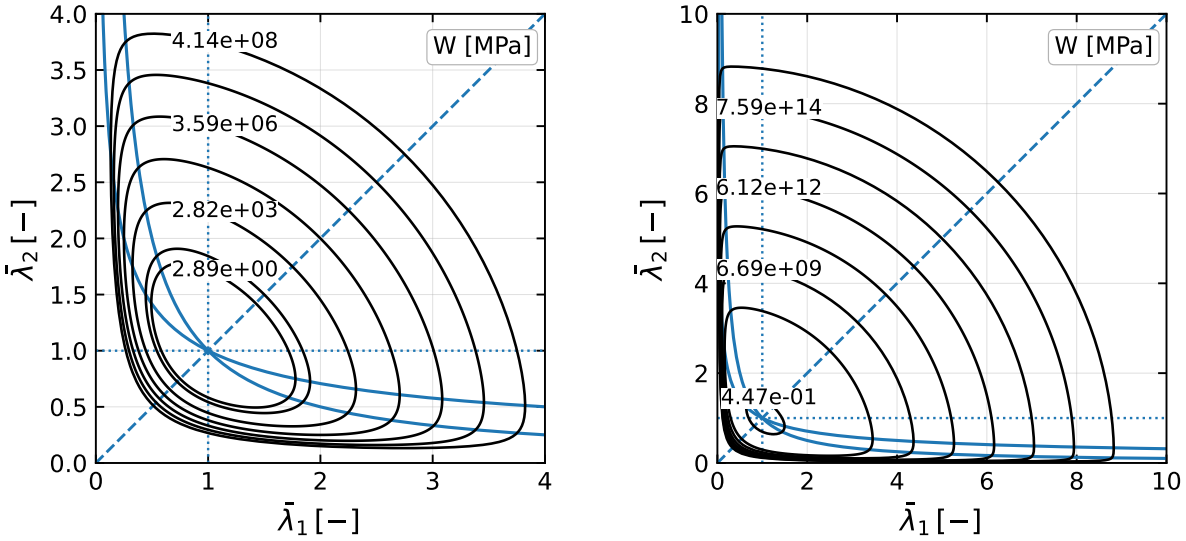
Na rysunku 6.7 można zauważyć, że wszystkie trzy wersje modelu Lasso dobrze odwzorowują dane eksperymentalne w zmierzonym zakresie wydłużeń. Ponadto małe współczynniki

przy wysokich potęgach niezmienników, które mogły być mylnie zinterpretowane jako artefakty numeryczne wydają się nie bez znaczenia fizycznego. Nachylenie przewidywanych krzywych na granicy zakresu danych Treloara sugerują, że poprawiają one zachowanie modelu w ekstrapolacji.

Dla kompletności zestawu wyników w tabeli 6.6 zestawiono metryki wersji modeli wyznaczonych metodą regresji Lasso, a na rysunku 6.8 przedstawiono wykresy warstwicowe funkcji jednostkowej energii sprężystości dla modelu v1.0.1.3, których przebiegi wyglądają poprawnie.

Tabela 6.6. Metryki testowe modeli hipersprężystych wyznaczonych metodą regresji Lasso w porównaniu z modelem Bidermana — uszeregowane od najlepszego do najgorszego

Wersja	**Model**	**R²(test)**	**MSE(test) [MPa]**
v1.0.1.3	LassoCV ($\alpha = 3.41 \times 10^{-3}$)	0.998	0.004
v1.0.1.1	Lasso ($\alpha = 10^{-2}$)	0.998	0.004
v1.0.0.4	Biderman	0.997	0.008
v1.0.1.2	Lasso ($\alpha = 10^{-1}$)	0.986	0.035



(a) Mały zakres deformacj

(b) Duży zakres deformacji

Rysunek 6.8. Wykresy warstwicowe funkcji jednostkowej energii sprężystości $W(\bar{\lambda}_1, \bar{\lambda}_2)$ w przypadku deformacji izochorycznej dla modelu LassoCV (v1.0.1.3). Niebieskimi liniami zaznaczono krzywe odpowiadające podstawowym testom doświadczalnym: jednoosiowemu rozciąganiu, dwuosiowemu rozciąganiu oraz czystemu ścinaniu, a linią przerywaną — przekątną $\bar{\lambda}_1 = \bar{\lambda}_2$.

6.4.3. Rozszerzenie bazy funkcyjnej

Jak dotąd baza funkcyjna obejmowała jedynie wyrażenia $(\bar{I}_1 - 3)^i(\bar{I}_2 - 3)^j$ inspirowane klasycznym wielomianem Rivlina–Saundersa. Takie założenie bazy stanowi problem, gdyż jak pokazano w sekcji 6.4.1 (warstwice modelu Mooney–Rivlina ($n = 2$)) tak zdefiniowana baza nie gwarantuje spełnienia warunku poliwypukłości funkcji energii sprężystości. Hartmann i Neff [120] oraz Jemioło [71], udowodnili, że klasyczne człony mieszane $(\bar{I}_1 - 3)^i(\bar{I}_2 - 3)^j$ z $i, j \geq 1$ nie są poliwypukłe. W rzeczywistości nie są nawet funkcjami wypukłymi I rzędu. Co więcej, człony $(\bar{I}_2 - 3)^j$ po multiplikatywnym rozkładzie objętościowo-izochorycznym tensora gradientu deformacji również nie zapewniają poliwypukłości funkcji JES ze względu na niewłaściwy dobór wykładników.

W tej sekcji postanowiono rozszerzyć bazę funkcyjną o szczególny przypadek poliwypukłych członów zaproponowanych przez Demiraya [113] (por. 5.1), poprawiając jednocześnie wykładniki stojące przy drugim niezmienniku deformacji izochorycznej [71]. Podsumowując, człony części izochorycznej obejmą:

$$\varphi_1^{(i)}(\bar{I}_1) = C_{i0}\left(\bar{I}_1 - 3\right)^i, \qquad i = 1, \ldots, N, \tag{6.16}$$

$$\varphi_2^{(i)}(\bar{I}_2) = \tilde{C}_{0i}\left(\bar{I}_2^{3/2} - 3\sqrt{3}\right)^i, \qquad i = 1, \ldots, N, \tag{6.17}$$

$$\varphi_3^{(k)}(\bar{I}_1) = \beta_{k0}\left[\exp\Big(\alpha_{k0}\left(\bar{I}_1 - 3\right)\Big) - 1\right], \qquad k = 1, \ldots, K, \tag{6.18}$$

$$\varphi_4^{(k)}(\bar{I}_2) = \tilde{\beta}_{0k}\left[\exp\Big(\alpha_{0k}\left(\bar{I}_2^{3/2} - 3\sqrt{3}\right)\Big) - 1\right], \qquad k = 1, \ldots, K. \tag{6.19}$$

Elementy $\varphi_3^{(k)}$ i $\varphi_4^{(k)}$ są poliwypukłymi uogólnieniami modelu Demiraya [113]. Zastosowano je do obydwu niezmienników izochorycznych. Zamiast ręcznie ustawiać parametry skalowania $\alpha_k^{(1)}$ i $\alpha_k^{(2)}$ oblicza się je automatycznie na podstawie danych treningowych. Dla każdego niezmiennika określa się najpierw maksymalny zakres argumentów wykładniczych:

$$A_{10} = \max_{\text{training data}}\left(\bar{I}_1 - 3\right), \qquad A_{01} = \max_{\text{training data}}\left(\bar{I}_2^{3/2} - 3\sqrt{3}\right). \tag{6.20}$$

Gdy już mamy te wartości, ustalamy podstawowe współczynniki skalowania w następujący sposób:

$$\alpha_{10} = \frac{a_{\max}}{A_{10}}, \qquad \alpha_{01} = \frac{a_{\max}}{A_{01}}. \tag{6.21}$$

Tutaj $a_{\max}$ jest z góry zdefiniowanym hiperparametrem, który reprezentuje najwyższą wartość argumentu wykładnika w naszym zbiorze szkoleniowym. Ustalono $a_{\max}$ na 5, co

odpowiada wartości około 148 przy użyciu funkcji wykładniczej. Należy zauważyć, że bez zabiegu skalowania wyrażenia wykładnicze $\varphi_3^{(k)}$ i $\varphi_4^{(k)}$ stają się niestabilne numerycznie. Dla przykładu, w próbie dwuosiowej (BT), gdzie $\lambda_1 = \lambda_2 = \lambda$, drugi niezmiennik izochoryczny jest definiowany jako $\bar{I}_2 = 2\lambda^{-2} + \lambda^4$, który rośnie wraz z λ^4. W ostatnim punkcie pomiarowym dla Treloara, gdy $\lambda \approx 4$, $\bar{I}_2$ przyjmuje wartość około 256. Jeśli podniesiemy tę wartość do potęgi 3/2 otrzymamy wartość powyżej 4000, która jako argument funkcji exp przekroczy maksymalny zakres formatu zmiennoprzecinkowego o ponad tysiąc rzędów wielkości!

W przypadkach, gdy k jest większe od 1, zastosowano liniowo malejące współczynniki:

$$\alpha_{k0} = \frac{\alpha_{10}}{k}, \qquad \alpha_{0k} = \frac{\alpha_{01}}{k}, \qquad k = 1, \ldots, K, \tag{6.22}$$

generując K cech o stopniowo słabnącej nieliniowości wykładniczej. Ponadto wprowadzono warunek nieujemności wszystkich współczynników poprzez ograniczenie `positive=True` w regresji LASSO w pakiecie `scikit-learn`.

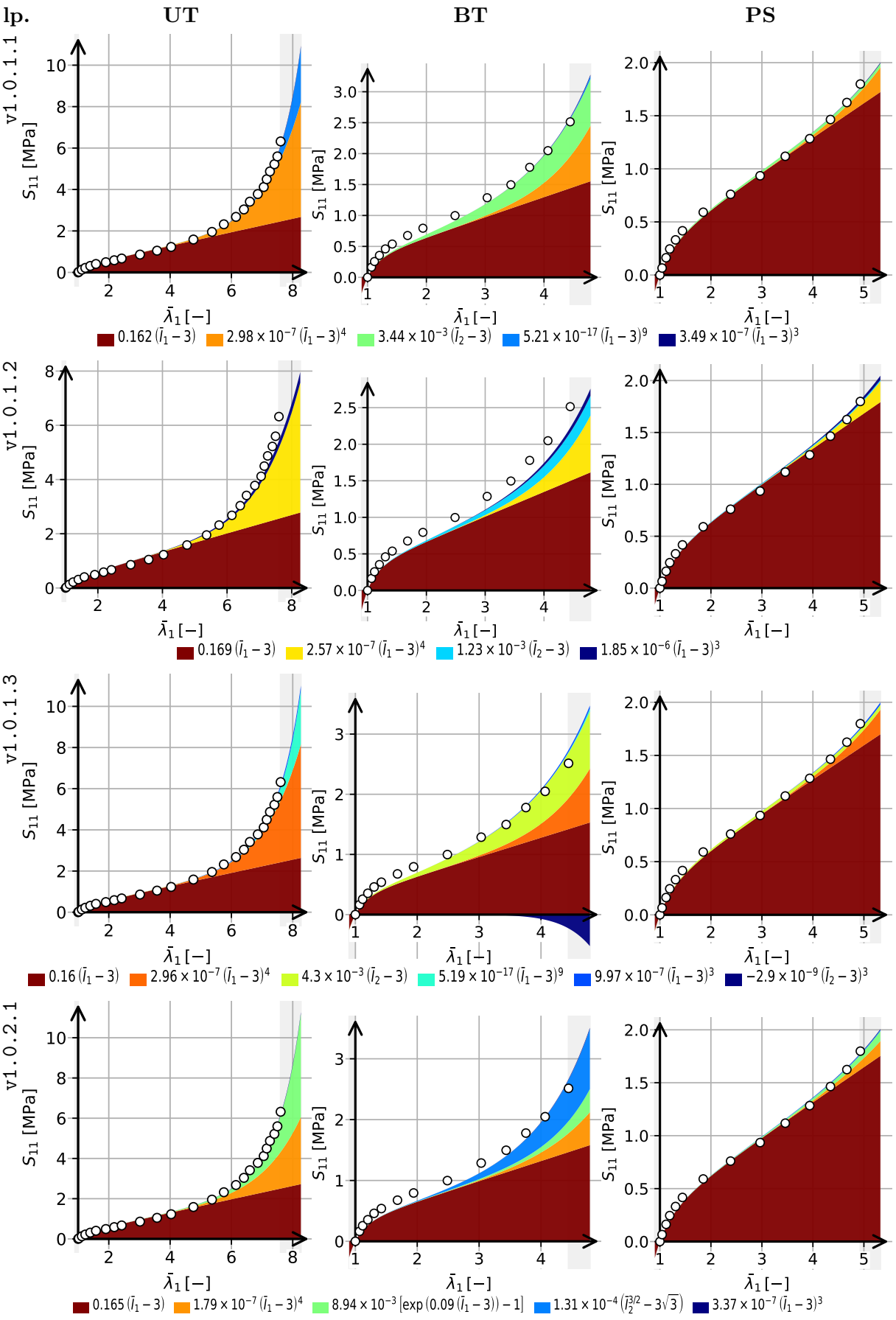
Bazę dodatkowo łatwo można rozszerzać o poliwypukłe człony objętościowe takie jak $-\ln J$, $J - \ln J$, $J^2 - 2\ln J + 4(\ln J)^2$ czy $(J^{2p} + J^{-2p} - 2)^k$ (por. 5.1 i [120]). W niniejszej pracy ze względu na nieściśliwy charakter danych Treloara ten etap został pominięty.

Model osiągnął metryki testowe $R^2 = 0.998$ i MSE $= 0.006$ MPa^2, porównywalne z najlepszymi z poprzedniej sekcji. Jego parametry zestawiono w tabeli 6.7.

Tabela 6.7. Parametry modelu z wyrażeniami Demiraya [113] wyznaczone metodą LassoCV na danych Treloara [33] dla $K = 3$ i $N = 9$

Wersja	**Model**	**Parametry**
v1.0.2.1	z wyrażeniami Demiraya	$C_{10} = 0.1649$
		$C_{30} = 3.365 \times 10^{-7}$
		$C_{40} = 1.793 \times 10^{-7}$
		$\tilde{C}_{01} = 1.307 \times 10^{-4}$
		$\beta_{10} = 8.938 \times 10^{-3}$, $\alpha_{10} = 9.074 \times 10^{-2}$

Rysunek 6.9 pokazuje jak poszczególne człony funkcji JES wpływają na naprężenie Pioli-Kirchhoffa II rodzaju dla czterech wersji modeli i trzech typów deformacji.



Rysunek 6.9. Wkład poszczególnych członów funkcji JES do natężenia $S_{11}(\bar{\lambda}_1)$ dla czterech wariantów modelu

W każdym przypadku liniowy człon $C_{10}(\bar{I}_1 - 3)$ ma największy wpływ na odpowiedź materiału, szczególnie w zakresie małych odkształceń. Wszystkie pozostałe człony stanowią poprawki, które mają coraz większy wpływ przy rosnących wydłużeniach.

Ujemny wkład (granatowy obszar na wykresie dwuosiowego rozciągania) pochodzący z wyrażenia $-2.9 \times 10^{-9}(\bar{I}_2 - 3)^3$, w wersji v1.0.1.3 sugeruje błąd w założeniu poliwypukłości. Mimo że wykresy warstwicowe 6.8 wyglądają poprawnie, to dopiero wykresy wkładu poszczególnych członów pozwalają sobie wyobrazić, że dany człon może być dominujący w pewnym dużym zakresie deformacji, choć w przypadku elastomeru — pewnie już po przekroczeniu granicy wytrzymałości na rozciąganie. W wersji v1.0.2.1 poliwypukłość została zagwarantowana analitycznie przez wprowadzenie ograniczenia nieujemności wszystkich współczynników.

Wersja v1.0.2.1 została wzbogacona o wyrażenia Demiraya. eksponencjalny człon odpowiada za charakterystyczne usztywnienie elastomeru w dużych deformacjach. Klasyczne wielomianowe człony nie wykazują tego efektu aż do uwzględnienia wyrazów wysokiego rzędu.

W każdym z modeli i przypadków obciążeń wpływ wyrażeń z pierwszym niezmiennikiem izochorycznym był większy niż z drugim, szczególnie w przypadku próby jednoosiowej. Natomiast w sytuacji dwuosiowego rozciągania jest on znaczny, co potwierdza słuszność włączenia go do bazy i poprawne wskazania macierzy korelacji z rys. 6.3.

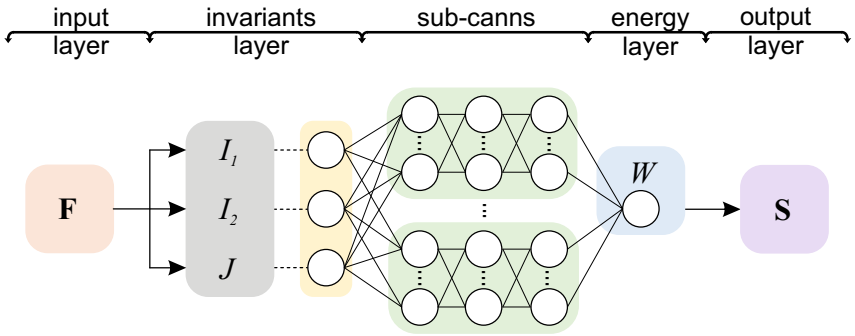
6.5. Wyznaczenie modeli konstytutywnych przy wykorzystaniu konstytutywnej sieci neuronowej

Niniejsza sekcja przedstawia opis implementacji konstytutywnej sieci neuronowej hipersprężystych materiałów nieściśliwych i izotropowych. Jej kod podzielono na trzy moduły, w którym `continuum.py` odpowiada za wzory z mechaniki ośrodków ciągłych, w module `models.py` znajduje się architektura sieci neuronowej, a `train.py`, jak sama nazwa wskazuje, jest skryptem treningowym.

Rozwiązanie obejmuje trzy podstawowe testy wytrzymałościowe opisane w 5.6.4. Tak naprawdę moduł `continuum.py` to wierne przepisanie wzorów (5.55)-(5.60) i jest pod tym względem uboższy od modelu regresji liniowej. To, na co naprawdę warto zwrócić uwagę w module `continuum.py`, to uzyskiwanie pochodnych jednostkowej energii sprężystości po niezmiennikach deformacji. Nie zapisuje się na nie jawnych wzorów, a różniczkuje

automatycznie, dzięki czemu trafiają one do grafu obliczeniowego. W konsekwencji funkcja straty może propagować te gradienty aż do wag CANN.

Architekturą sieci neuronowej jest sieć PFNN (Parallel Forward Neural Network) z rygorystycznymi warunkami konstruowania każdej podsieci wynikającymi ze spełnienia wymogów mechaniki ośrodków ciągłych. Jednym z tych warunków wpływających na architekturę jest ograniczenie wszystkich wag sieci do nieujemnych. Ogólna struktura sieci PFNN została opisana na str. 163 przy okazji omówienia wariantu W3 w podrozdziale 7.4.3. Sposób konstruowania podsieci jest uzależniony od tego, jakimi funkcjami zależnymi od niezmienników chcemy aproksymować naszą funkcję JES. Powinny być one złożeniem z szeregu klasycznych modeli hipersprężystości. Dzięki temu sieć nie działa jak *czarna skrzynka*, a po wytrenowaniu ujawnia, które człony są aktywne i z jakich modeli pochodzą. Ogólny schemat sieci neuronowej przedstawiono na rysunku 6.10. Dane



Rysunek 6.10. Ogólny schemat konstytutywnej sieci neuronowej

treningowe zostały zinterpolowane w taki sposób, aby każdy z trzech podstawowych testów wytrzymałościowych składał się z takiej samej liczby punktów $n/3$. Funkcja straty jest błędem średniokwadratowym wszystkich punktach treningowych i została uzupełniona o składnik regularyzacyjny L_2:

$$L(\underline{w},\, \boldsymbol{F}) = \frac{1}{n}\sum_{i=1}^{n}\left\|\boldsymbol{S}(\boldsymbol{F}_i,\, \underline{w}) - \hat{\boldsymbol{S}}_i\right\|^2 \;+\; \alpha\, \underline{w}^{\top}\underline{w} \tag{6.23}$$

gdzie S to pomierzone naprężenia Pioli-Kirchhoffa I rodzaju, a α współczynnikiem regularyzacji L_2 równym $L_2 = 10^{-3}$. Minimalizację prowadzono algorytmem Adam z krokiem uczenia $\eta = 10^{-3}$. W pętli treningowej zastosowano mechanizm wczesnego zatrzymania po 1000 epokach bez poprawy, a liczbę epok dobrano tak, aby on zadziałał. Uważnego czytelnika może zastanowić, *dlaczego w tym przypadku kara L_2 wygasza wagi, skoro to jest cechą regularyzacji L_1?*. Regularyzacja ciągnie każdą wagę w kierunku zera i jeśli krok

optymalizatora ją przekroczy, waga zostanie tam przycięta i ze względu na zastosowanie nieujemnych wag algorytm nie pozwoli już jej wrócić.

6.5.1. Wariant z członami liniowymi, potęgowymi i eksponencjalnymi

Jak już wspomniano wcześniej sieć CANN jest uogólnieniem szeregu klasycznych modeli, w tym przypadku są to modele neo-Hooke'a, Yeoha i Demiraya por. 6.2. Pierwsza warstwa każdej z dwóch podsieci generuje potęgi niezmiennika deformacji izochorycznej $\left(\bar{I}_i - 3\right)$ – $(\circ)$ oraz $(\circ)^2$, natomiast druga stosuje do nich dwie funkcje nietypowe aktywacji: tożsamość $(\circ)$ oraz wykładniczą $\exp(\circ) - 1$. Zazwyczaj nakładane funkcje aktywacji na każde wyjście w ukrytej warstwy są identyczne – tu sytuacja jest odmienna. Razem mamy cztery bazy funkcyjne na każdy niezmiennik, a całą sieć możemy zapisać jako:

$$\begin{aligned} W\left(\bar{I}_1, \bar{I}_2\right) = {} & w_{2,1}\, w_{1,1}\left(\bar{I}_1 - 3\right) + w_{2,2}\Big[\exp\Big(w_{1,2}\left(\bar{I}_1 - 3\right)\Big) - 1\Big] \\ & + w_{2,3}\, w_{1,3}\left(\bar{I}_1 - 3\right)^2 + w_{2,4}\Big[\exp\Big(w_{1,4}\left(\bar{I}_1 - 3\right)^2\Big) - 1\Big] \\ & + w_{2,5}\, w_{1,5}\left(\bar{I}_2 - 3\right) + w_{2,6}\Big[\exp\Big(w_{1,6}\left(\bar{I}_2 - 3\right)\Big) - 1\Big] \\ & + w_{2,7}\, w_{1,7}\left(\bar{I}_2 - 3\right)^2 + w_{2,8}\Big[\exp\Big(w_{1,8}\left(\bar{I}_2 - 3\right)^2\Big) - 1\Big]. \end{aligned} \tag{6.24}$$

Architektura jest odtworzeniem drugiego przykładu w rozdziale 5 z [15]. Inicjalizacja wag wielomianowych opiera się na rozkładzie $\mathcal{N}(0,\, 0.3)$, a wag wykładniczych na $\mathcal{U}(0,\, 10^{-5})$ – człony wykładnicze startują bliskie zeru przez co model powinien preferować prostsze wielomianowe wyrażenia algebraiczne.

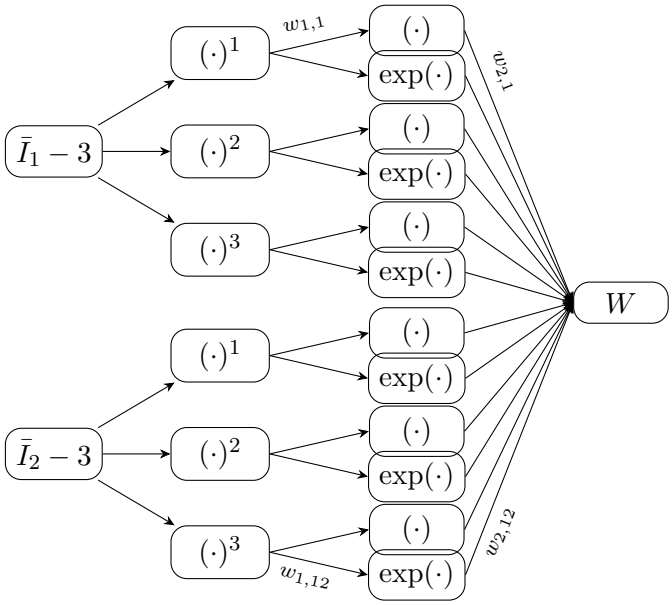
6.5.2. Rozszerzenie o człony sześcienne i normalizacja niezmienników

Jak już sugeruje nam tytuł, w niniejszym przykładzie rozszerzono architekturę o człony sześcienne. Pozwala to uwzględnić wszystkie człony modelu Yeoha. Ten zabieg ujawnił problem numeryczny znany nam już z 6.4.3, por. (6.14). W tym przypadku również wejściowe wyrażenia z niezmiennikami znormalizowano, co również finalnie wymagało przeskalowania wag. W implementacji sieć oczywiście operuje na znormalizowanych wejściach, ale rozszerzoną wersję architektury, tak by pozostała bezpośrednią reprezentacją wyuczonego prawa

materiałowego, a także była zgodna z (6.24), przedstawia się po powrotnym skalowaniu:

$$\begin{aligned} W(\bar{I}_1, \bar{I}_2) = {} & w_{2,1}\, w_{1,1}\, (\bar{I}_1 - 3) + w_{2,2}\Big[\exp\Big(w_{1,2}\, (\bar{I}_1 - 3)\Big) - 1\Big] \\ & + w_{2,3}\, w_{1,3}\, (\bar{I}_1 - 3)^2 + w_{2,4}\Big[\exp\Big(w_{1,4}\, (\bar{I}_1 - 3)^2\Big) - 1\Big] \\ & + w_{2,5}\, w_{1,5}\, (\bar{I}_1 - 3)^3 + w_{2,6}\Big[\exp\Big(w_{1,6}\, (\bar{I}_1 - 3)^3\Big) - 1\Big] \\ & + w_{2,7}\, w_{1,7}\, (\bar{I}_2 - 3) + w_{2,8}\Big[\exp\Big(w_{1,8}\, (\bar{I}_2 - 3)\Big) - 1\Big] \\ & + w_{2,9}\, w_{1,9}\, (\bar{I}_2 - 3)^2 + w_{2,10}\Big[\exp\Big(w_{1,10}\, (\bar{I}_2 - 3)^2\Big) - 1\Big] \\ & + w_{2,11}\, w_{1,11}\, (\bar{I}_2 - 3)^3 + w_{2,12}\Big[\exp\Big(w_{1,12}\, (\bar{I}_2 - 3)^3\Big) - 1\Big]. \end{aligned} \tag{6.25}$$

Ze względu na mniejsze wartości wejść do sieci zwiększono wartości w inicjalizacji wag wykładniczych do rozkładu $\mathcal{U}(0,\, 10^{-3})$. Kod realizujący architekturę sieci CANN użytą w wersji v1.1.1.1 znajduje się w zał. 6, a jej schemat blokowy możemy znaleźć na rysunku poniżej:



Rysunek 6.11. Schemat podsieci architektury CANN w wersji v1.1.1.1

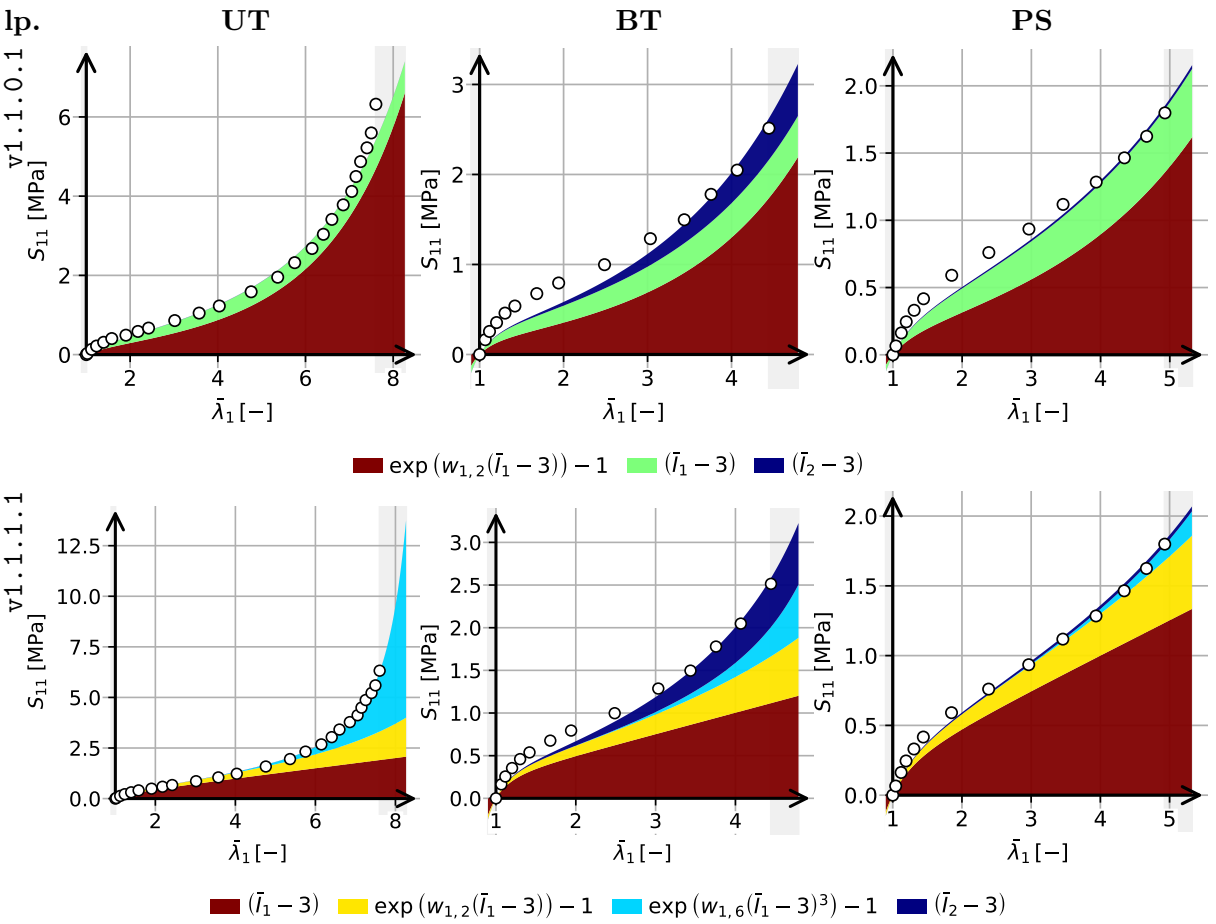
6.5.3. Porównanie wyników

W niniejszej sekcji znajdą się wyniki dopasowania do wszystkich podstawowych testów wytrzymałościowych jednocześnie. Należy zaznaczyć, że kod został zaimplementowany od podstaw w PyTorch. W związku z tym zaprezentowane wartości i wykresy dla wariantu v1.1.0.1 mogą nieznacznie różnić się od wyników uzyskanych w pracy [15]. Parametry

modeli z rozdziałów 6.5.1 i 6.5.2 zostały przedstawione w tab. 6.8, a wkład poszczególnych członów JES w naprężenia Pioli-Kirchhoffa I rodzaju zaprezentowano na rys. 6.12.

Tabela 6.8. Modele CANN – osteteczne postaci funkcji jednostkowej energii sprężystości

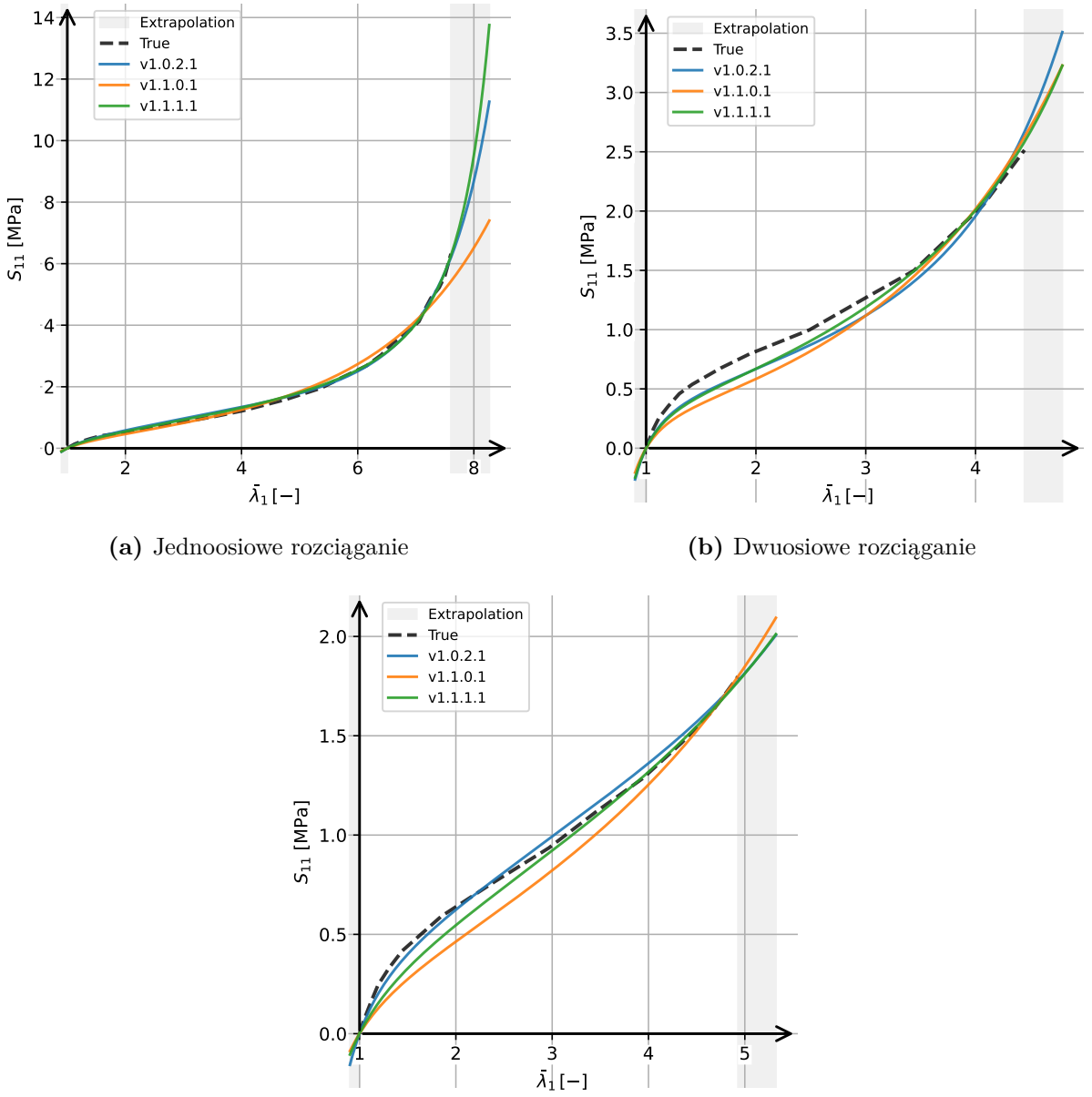
Wersja	**Postać JES**	**Parametry**
v1.1.0.1	$W(\bar{I}_1,\bar{I}_2) = C_{10}(\bar{I}_1-3) + C_{01}(\bar{I}_2-3) + \beta_{10}\Big[\exp\Big(\alpha_{10}(\bar{I}_1-3)\Big)-1\Big]$	$C_{10} = 4.777\times10^{-2}$ $C_{01} = 2.631\times10^{-3}$ $\alpha_{10} = 2.201\times10^{-2}$ $\beta_{10} = 3.219$
v1.1.1.1	$W(\bar{I}_1,\bar{I}_2) = C_{10}(\bar{I}_1-3) + C_{01}(\bar{I}_2-3) + \beta_{10}\Big[\exp\Big(\alpha_{10}(\bar{I}_1-3)\Big)-1\Big] + \beta_{30}\Big[\exp\Big(\alpha_{30}(\bar{I}_1-3)^3\Big)-1\Big]$	$C_{10} = 0.1255$ $C_{01} = 3.278\times10^{-3}$ $\alpha_{10} = 2.201\times10^{-2}$ $\beta_{10} = 1.255$ $\alpha_{30} = 6.681\times10^{-6}$ $\beta_{30} = 1.043$



Rysunek 6.12. Wkład poszczególnych członów funkcji JES do naprężenia $S_{11}(\bar{\lambda}_1)$ dla dwóch wariantów modelu CANN

Pomimo zbliżonych form postaci JES do 6.7, rozwiązania z CANN charakteryzują się innym rozkładem (por. z wersją v1.0.2.1 na rys. 6.9). Regresja Lasso preferuje człony z uogólnionego modelu Mooneya-Rivlina, natomiast sieć rozkłada wkład między człony w sposób bardziej równomierny. Wynika to z kilku rzeczy – w regresji współczynniki przy funkcji eksponencjalnej zostały wyznaczone *a priori* przed wykonaniem optymalizacji, sieć neuronowa nie miała dostępu do wyrażeń wykładniczych stopnia wyższego od 3, a i sposób regularyzacji był odmienny. Regularyzacja L_1 zastosowana w LassoCV naturalnie wymusza zerowanie się współczynników, natomiast regresja grzbietowa L_2 jedynie proporcjonalnie zmniejsza ich wartości - dopiero ograniczenie nieujemności wag po kroku optymalizatora to wykonuje.

Wykresy z trzech podstawowych testów wytrzymałościowych zestawiono z modelem regresji liniowej z członami Demiraya z rozdziału 6.4.3 i przedstawiono na rys.6.13, a metryki testowe zamieszczono w tab.6.9.



(a) Jednoosiowe rozciąganie

(b) Dwuosiowe rozciąganie

(c) Czyste ścinanie

Rysunek 6.13. Wykresy S_{11} względem wydłużenia λ_1: porównanie modeli uzyskanych konstytutywnymi sieciami neuronowymi z danymi eksperymentalnymi Treloara [33] oraz modelem regresji liniowej (v1.0.2.1)

Tabela 6.9. Współczynniki determinacji R^2

Wersja	**Model**	R^2_{UT}	R^2_{PS}	R^2_{BT}	R^2_{combined}
v1.0.2.1	Regresja liniowa (Demiray)	0.998	0.996	0.978	0.998
v1.1.0.1	CANN – Linka (8 członów)	0.986	0.980	0.961	0.986
v1.1.1.1	CANN – rozszerzony (12 członów)	0.999	0.995	0.984	0.998

Wprowadzenie normalizacji danych wejściowych w wersji v1.1.1.1 nie tylko rozwiązało problem związany z przekroczeniem zakresu zmiennoprzecinkowego, co w ogóle umożliwiło rozszerzenie rozwiązania o dodatkowe człony funkcji JES, ale też poprawiło wyniki w każdym z trzech podstawowych testów wytrzymałościowych. Warto podkreślić, że standaryzacja czy normalizacja zmiennych wejściowych przy rozwiązywaniu zadań uczenia maszynowego wrażliwych na skalę danych, takich jak regresja liniowa czy sieci neuronowe, jest procesem powszechnym i zalecanym – nie faworyzuje zmiennych o większym zakresie, a w sieciach neuronowych – przyspiesza zbieżność i notabene niektóre z funkcji aktywacji, jak np. tanh czy funkcja sigmoidalna są najefektywniejsze (mają najwyższą pochodną) w zakresie bliskim zera — choć tu, ze względu na zastosowanie ich niestandardowych wersji, problem nie występuje.

Wartości współczynnika determinacji R^2 dla trzech innych modeli z tab. 6.9 są bardzo wysokie, ale nieznacznie się różnią. Mimo pozornie niewielkich różnic między modelami, przy takich rezultatach współczynnik determinacji bliższy jedności będzie prawdopodobnie lepiej przewidywał również wartości z końca zakresu, co będzie wpływało również na nachylenie krzywych w tych miejscach. Różnice są widoczne na wykresach 6.13, szczególnie poza zakresem danych eksperymentalnych w teście jednoosiowego rozciągania. Model v1.1.0.1 wyraźnie odbiega od krzywej referencyjnej, podczas gdy pozostałe zachowują się poprawnie.

6.6. Wnioski

Warto zauważyć, że wartości współczynnika determinacji R^2 z przedziału 0.97–0.999 byłoby uważane za dobre w kontekście klasycznego zastosowania algorytmów uczenia maszynowego. Niemniej jednak, kiedy modelujemy prawa konstytutywne hipersprężystości, używanie go może być niewystarczające – zdolność do ekstrapolacji danych jest kluczowa, czego R^2 nie zapewnia. Zasadne mogłoby być sprawdzanie zgodności wartości nachylenia krzywych *naprężenie–wydłużenie* na krańcach przedziału (albo i w całym obszarze [123]) i uwzględnianie ich w funkcjach kosztu i metrykach. Podejście to nazywa się treningiem Sobolewa i zostało wdrożone w pracy [124] do materiałów sprężysto-plastycznych. Co podkreślano wcześniej, współczynnik R^2 bliższy jedności będzie prawdopodobnie lepiej przewidywał dane również na krańcach przedziału, a co za tym idzie – ekstrapolował testy.

Żaden z modeli nie wpasował się tak dobrze w stan dwuosiowego rozciągania jak pozostałe rodzaje obciążenia, mimo normalizacji zmiennych wejściowych od modelu wersji v1.0.1.1 wzwyż — nie ma zatem sposobności, aby wartości z testu jednoosiowego rozciągania dominowały nad pozostałymi testami i modele preferowałyby dopasowanie się właśnie do tego stanu. Jest to stan najbardziej czuły na drugi niezmiennik deformacji izochorycznej i powodem może być zbyt uboga baza funkcyjna.

Udowodniono, że zastosowanie modeli wyłącznie zależnych od pierwszego niezmiennika deformacji izochorycznej (neo-Hooke, Yeoh) jest niewystarczające – wszystkie modele, które w swojej bazie funkcyjnej miały wyrażenia z $\bar{I}_2$ uwzględniały je.

W zadaniach z regresji otrzymano odmienne modele materiałowe niż w sieciach neuronowych, mimo że oba są poprawne. Regresja Lasso preferowała człony wielomianowe, a sieć neuronowa bardziej równomiernie rozłożyła wkłady poszczególnych funkcji bazowych.

Niniejszy rozdział wnosi wkład do literatury i jest oryginalny w kilku aspektach. Pierwszym z nich jest cały moduł dotyczący liniowej regresji – umożliwia on analizowanie pełnych, różnorodnych stanów deformacji oraz wprowadza pojęcie macierzy naprężeniowej cech, której szczegółowe działanie przedstawiono w studium przypadku. Ponadto wprowadzono modele, łącząc je zgodnie z pracą [120], wprowadzając właściwą potęgę niezmiennika $\bar{I}_2^3/2$. Co więcej, w obu modelach, liniowej regresji i sieci neuronowej, wejścia znormalizowano, co wymagało zastosowania odwrotności normalizacji do współczynników materiałowych. W liniowej regresji było to możliwe dzięki zastosowaniu pełnego procesu przetwarzania danych (ang. *pipeline*).

7. Sieci neuronowe oparte na fizyce (PINN) w zagadnieniach brzegowych mechaniki ośrodków ciągłych

7.1. Wprowadzenie

Jak już wcześniej wspomniano, podstawowa idea fizycznie informowanych sieci neuronowych *Physics-Informed Neural Networks* (PINN) [2] polega na zastąpieniu klasycznych metod dyskretyzacji, takich jak np. MES [9] czy MRS siecią neuronową, która aproksymuje rozwiązanie równania. Kluczowym krokiem jest wówczas wymuszenie, aby sieć minimalizowała resztę równania różniczkowego (tzw. *PDE residual*). W porównaniu z tradycyjnymi metodami opartymi na siatce numerycznej, podejście to jest w pełni bezsiatkowe i wykorzystuje automatyczne różniczkowanie [42], które pozwala wyznaczać pochodne dokładnie i eliminuje błąd dyskretyzacji pochodnych typowy dla różniczkowania numerycznego. Sieć pozostaje jednak aproksymacją rozwiązania, więc błąd przybliżenia nie znika. Atrakcyjną cechą sieci PINN jest to, że mogą być one wykorzystywane do rozwiązywania problemów odwrotnych (*inverse problems*) [2], [125]–[128]. Z czasem PINN zostały rozszerzone w celu rozwiązywania równań całkowo-różniczkowych, równań różniczkowych cząstkowych oraz stochastycznych równań różniczkowych [2], [75], [129], [130].

Na bazie tych idei powstała biblioteka `DeepXDE` (ang. *Deep Learning Library for Differential Equations*) [131], napisana w języku Python. DeepXDE obsługuje złożone geometrie oparte na metodzie konstruktywnej geometrii bryłowej (CSG [131], *Constructive Solid Geometry*), umożliwia łatwą definicję warunków początkowych i brzegowych oraz zapewnia elastyczne monitorowanie procesu uczenia za pomocą systemu wywołań zwrotnych (*callbacks*).

Biblioteka DeepXDE implementuje pełny algorytm sieci PINN, obejmujący definiowanie dziedziny obliczeniowej, formułowanie równania różniczkowego, określenie warunków brzegowych i początkowych, budowę sieci neuronowej, kompilację modelu oraz jego trening. W kolejnych podrozdziałach przedstawiono schemat algorytmu PINN, sposób jego implementacji w bibliotece DeepXDE oraz przykładowe zastosowania w analizie materiałów i identyfikacji parametrów konstytutywnych.

7.2. Rozwiązanie jednowymiarowego równania Poissona

W niniejszym podrozdziale przedstawiono przykład rozwiązania jednowymiarowego równania Poissona z wykorzystaniem Physics-Informed Neural Networks (PINNs). Celem tego przykładu jest wprowadzenie czytelnika w podstawowe etapy konstruowania i uczenia modeli PINN oraz zaprezentowanie biblioteki DeepXDE jako narzędzia wspomagającego implementację tego typu metod. W szczególności pokazano, w jaki sposób biblioteka DeepXDE umożliwia sformułowanie składowych zadania obejmujących definicję dziedziny funkcji, narzucenie warunków brzegowych, zapis reszt równań różniczkowych jako składnika funkcji kosztu, dobór punktów treningowych i testowych oraz procedurę uczenia. Podejście zostało porównane z analogiczną implementacją w środowisku PyTorch, wymagającą jawnej realizacji wymienionych elementów. Takie porównanie pozwala z jednej strony zilustrować istotę metody PINN, a z drugiej — wskazać rolę biblioteki DeepXDE w redukcji złożoności implementacyjnej oraz w uporządkowaniu procesu prowadzenia obliczeń numerycznych.

Rozważmy prosty problem brzegowy (por. [132]):

$$\frac{d^2y(x)}{dx^2} + \pi^2 \sin(\pi x) = 0, \qquad x \in [-1; 1]. \tag{7.1}$$

z warunkami brzegowymi:

$$y(-1) = 0, \qquad y(1) = 0. \tag{7.2}$$

Rozwiązanie analityczne rozważanego problemu ma postać:

$$y(x) = \sin(\pi x). \tag{7.3}$$

Twierdzenie o uniwersalnej aproksymowalności [4], [40] stwierdza, że jednokierunkowa sieć neuronowa typu *feed-forward* z co najmniej jedną warstwą ukrytą, przy założeniu odpowiedniej funkcji aktywacji oraz skończonej liczby neuronów, może z dowolnie zadaną dokładnością przybliżać każdą funkcję ciągłą określoną na dziedzinie domkniętej i ograniczonej. Będziemy projektowali prostą sieć neuronową $y_{NN}(x)$ w celu aproksymacji rozwiązania $y(x)$:

$$y(x) \approx y_{NN}(x), \tag{7.4}$$

W klasycznym podejściu do uczenia maszynowego w uczeniu nadzorowanym [1], [43], model jest trenowany na zbiorze danych zawierającym pary wejście–wyjście. W przeci-

wieństwie do tego, sieci PINN (Physics-Informed Neural Networks) nie wymagają pełnego, klasycznego zbioru danych. Proces uczenia jest determinowany przez prawa fizyki opisujące układ, najczęściej w postaci równań różniczkowych. Źródłem danych są wówczas:

- równania bilansu [133], [134], które modelują zachowanie układu,
- warunki brzegowe i początkowe, które narzucają rozwiązaniu wartości na brzegu dziedziny i/lub w chwili początkowej.

W sieciach PINN równania bilansu pełnią rolę źródła nieskończonej liczby punktów danych. Równanie bilansu definiuje funkcje reszty, którą można obliczać w dowolnych punktach domeny. Jest to możliwe dzięki różniczkowalności sieci neuronowej $y_{NN}(x)$. Sieć neuronowa jest trenowana poprzez minimalizację reszt tych równań, a sama funkcja reszty w naszym przykładzie ma postać:

$$R(x) = \frac{d^2 y_{NN}(x)}{dx^2} + \pi^2 \sin(\pi x), \tag{7.5}$$

W naszym przykładzie składnikiem funkcji kosztu od niespełnienia równania bilansu będzie błąd średniokwadratowy z reszt na domenie w punktach treningowych $\{x_i\}_{i=1}^{N} \subset \Omega$:

$$L_{PDE} = \frac{1}{N} \sum_{i=1}^{N} R(x_i)^2 \;\approx\; \frac{1}{|\Omega|} \int_{\Omega} R(x)^2 \, dx. \tag{7.6}$$

Analogicznie do składnika L_{PDE}, warunki brzegowe (i/lub początkowe) dostarczają informacji o rozwiązaniu na brzegu $\partial\Omega$ (lub w chwili $t = t_0$). Dla warunku Dirichleta, narzucającego wartości rozwiązania, definiuje się resztę warunku brzegowego jako różnicę pomiędzy wartością zadaną a predykcją sieci:

$$B(x) = y_{NN}(x) - y_{BC}(x), \qquad x \in \partial\Omega, \tag{7.7}$$

gdzie $y_{BC}(x)$ oznacza wartość wymaganą przez warunek brzegowy.

W praktyce rozważa się często K różnych warunków brzegowych (lub grup warunków), z których każdy jest egzekwowany na własnym zbiorze punktów $\{x_j^{BC,k}\}_{j=1}^{N_{BC,k}} \subset \partial\Omega$. Dla k-tego warunku brzegowego odpowiadający mu składnik funkcji kosztu przyjmuje postać:

$$L_{BC,k} = \frac{1}{N_{BC,k}} \sum_{j=1}^{N_{BC,k}} \left(B_k(x_j^{BC,k}) \right)^2 = \frac{1}{N_{BC,k}} \sum_{j=1}^{N_{BC,k}} \left(y_{NN}(x_j^{BC,k}) - y_{BC,k}(x_j^{BC,k}) \right)^2. \tag{7.8}$$

Wówczas całkowita funkcja kosztu jest w ogólności sumą ważoną poszczególnych składników:

$$L = w_{PDE}\, L_{PDE} + \sum_{k=1}^{K} w_{BC,k}\, L_{BC,k}, \tag{7.9}$$

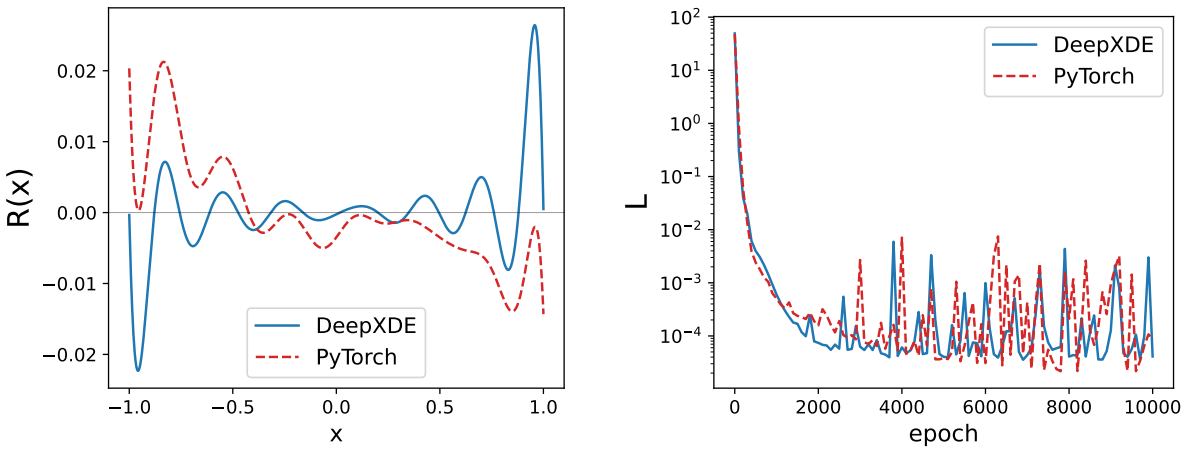
gdzie w_{PDE} jest wagą składnika równania (reszty PDE), natomiast $w_{BC,k}$ są wagami przypisanymi k-temu warunkowi brzegowemu (lub k-tej grupie warunków).

W załączniku 7 przedstawiono dwie równoważne implementacje powyższego zadania: z wykorzystaniem biblioteki DeepXDE oraz w postaci „czystej" implementacji w PyTorch.

W implementacji DeepXDE definicja reszty równania $R(x)$ jest realizowana w funkcji `pde(x, y)`, przy czym obliczenie pochodnej drugiego rzędu $\frac{\partial^2 y_{NN}}{\partial x^2}$ realizowane jest przez wywołanie `dde.grad.hessian(y, x)`. Dziedzina $\Omega = [-1, 1]$ jest zadana poprzez `dde.geometry.Interval(-1, 1)`, natomiast warunki brzegowe Dirichleta (7.2) są opisane obiektami `dde.icbc.DirichletBC` z funkcjami selekcji brzegu (`boundary_left`, `boundary_right`). Klasa `dde.data.PDE` odpowiada za konstrukcję zbioru punktów treningowych w domenie i na brzegu oraz za zestawienie składników funkcji kosztu. Procedura uczenia oraz monitorowanie przebiegu minimalizacji realizowane są przez wywołania `model.compile(...)` i `model.train(...)`.

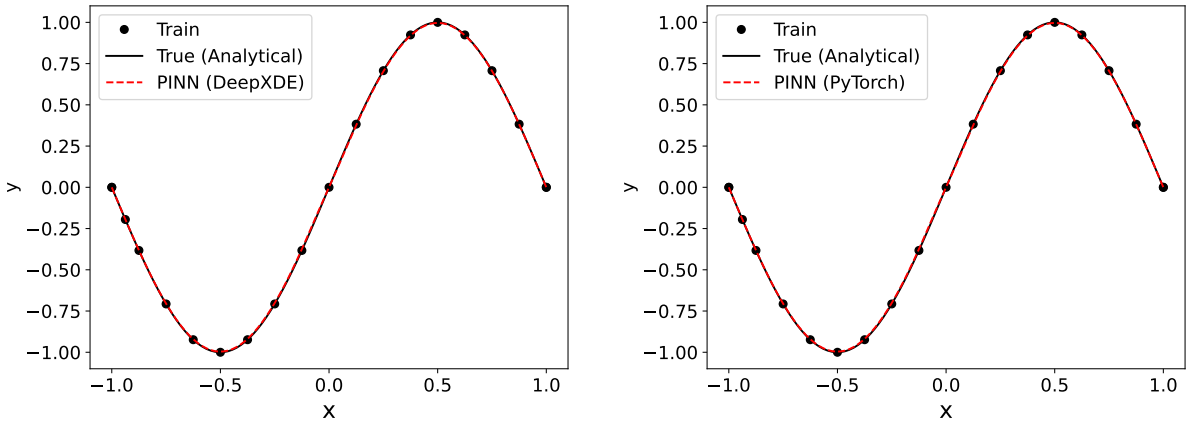
W implementacji PyTorch wszystkie powyższe elementy są zapisane jawnie. Reszta równania $R(x)$ jest obliczana poprzez wyznaczenie pochodnych $\frac{\partial y}{\partial x}$ i $\frac{\partial^2 y}{\partial x^2}$ przy użyciu `torch.autograd.grad`, a następnie jest złożona w funkcji `pde(model, x)`. Punkty z dziedziny $\{x_i\}$ są konstruowane jawnie (w tym przypadku przez `torch.linspace`), a warunki brzegowe reprezentowane są przez osobny zbiór punktów x_{BC} oraz wartości zadane y_{BC}. Funkcja `loss(...)` odpowiada za zbudowanie L_{PDE} i L_{BC} oraz ich sumowanie, natomiast pętla treningowa kontroluje aktualizację parametrów sieci i logowanie postępu.

W obu implementacjach przyjęto tę samą konfigurację: architekturę $[1, 50, 50, 50, 1]$ z aktywacją tanh i inicjalizacją Glorota, równomierne punkty kolokacyjne na $[-1, 1]$, optymalizator Adam ze stałym $lr = 10^{-3}$ przez 10^4 iteracji oraz koszt $L = L_{PDE} + L_{BC}$. Funkcje reszty $R(x)$ nie pokrywają się dokładnie, co nie wynika ze sformułowania zadania, lecz mają charakter numeryczny i stochastyczny. Obie sieci startują z niezależnie wylosowanych wag i wyznaczają drugą pochodną inną ścieżką różniczkowania (`torch.autograd.grad` wobec `dde.grad.hessian`), co przy stałym lr prowadzi do nieco innych, lecz równoważnych rozwiązań: $y(x) \approx \sin(\pi x)$ z resztą rzędu 10^{-2} i stratą 10^{-4}. Reszty równania i przebiegi minimalizacji funkcji kosztu zestawiono na rys. 7.1, a wyniki obu podejść na rys. 7.2.



(a) Reszta równania $R(x)$ (b) Strata testowa $L(\text{epoch})$

Rysunek 7.1. Porównanie reszty równania oraz przebiegu funkcji kosztu dla obu implementacji PINN



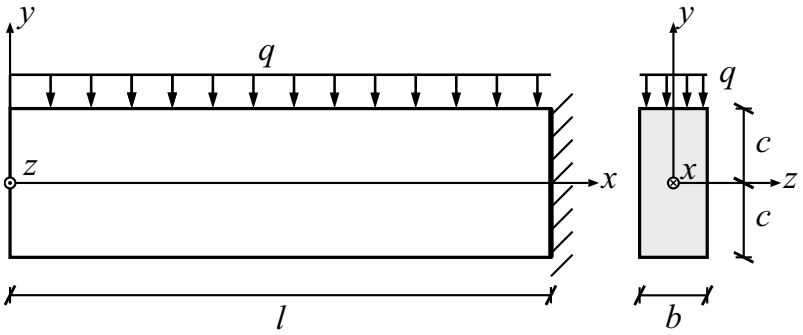
(a) DeepXDE (b) PyTorch

Rysunek 7.2. Porównanie rozwiązań równania Poissona 1D uzyskanych metodą PINN w dwóch implementacjach

Biblioteka DeepXDE zapewnia wyższy poziom abstrakcji, automatyzując definicję geometrii, warunków brzegowych i procedury uczenia, natomiast implementacja w „czystym" PyTorch wymaga jawnego zapisu pochodnych i pętli optymalizacji, oferując większą kontrolę nad próbkowaniem i wagowaniem składników funkcji kosztu. W związku z tym w kolejnych zadaniach wykorzystywana będzie biblioteka DeepXDE.

7.3. Belka wspornikowa obciążona równomiernie

W niniejszej części rozważono klasyczne zadanie zginania belki wspornikowej obciążonej równomiernie na całej długości. Model ten stanowi punkt odniesienia do dalszych porównań z tarczą w płaskim stanie naprężenia w ramach teorii sprężystości oraz do weryfikacji implementacji numerycznych PINN. Najpierw zadanie zostanie rozwiązane w ramach elementarnej teorii belek Eulera-Bernoulliego, tj. przy założeniu teorii małych przemieszczeń i odkształceń oraz pominięciu wpływu ścinania na krzywiznę (belka smukła). Ciężar własny zostaje pominięty. Rozwiązania analityczne w tej sekcji przyjęto na podstawie rękopisu do Wykładu z przedmiotu Teorii Sprężystości i Plastyczności [135].



Rysunek 7.3. Schemat zagadnienia brzegowego belki obciążonej równomiernie

Tabela 7.1. Przyjęte parametry

Parametr	Wartość
p	0.4 [kN/cm]
b	20.0 [cm]
$q = \frac{p}{b}$	0.02 [kN/cm^2]
E	30.0 [GPa]
ν	0.2
c	25.0 [cm]
l	200.0 [cm]

7.3.1. Rozwiązanie analityczne wg teorii belek

Wprowadzamy ugięcie w zmiennej wymiarowej jako $\mathsf{v}(x)$, a następnie przechodzimy na współrzędną bezwymiarową $\xi = x/l$. Ugięcie jako funkcję zmiennej bezwymiarowej możemy zapisać jako:

$$v(\xi) = \mathsf{v}(x)\Big|_{x=l\,\xi} = \mathsf{v}(l\,\xi), \qquad \xi \in [0,1]. \tag{7.10}$$

Korzystając z zależności $\frac{d}{dx} = \frac{1}{l}\frac{d}{d\xi}$, równanie ugięcia belki przyjmuje postać:

$$\frac{1}{l^4}\frac{d^4 v(\xi)}{d\xi^4} = \frac{p(l\,\xi)}{E J_z}, \qquad \text{gdzie} \qquad J_z = \frac{(2c)^3\, b}{12} = \frac{2}{3}\, b\, c^3. \tag{7.11}$$

W analizowanym przypadku mamy $p(x) = -p$, czyli $p(l\xi) = -p$ i równanie różniczkowe ugięcia belki przyjmuje postać:

$$\frac{d^4 v(\xi)}{d\xi^4} + \frac{3p\,l^4}{2E\,b\,c^3} = 0. \tag{7.12}$$

Wprowadźmy współczynnik skalujący ugięcie γ_{ref} oraz ugięcie bezwymiarowe $\hat{v}$:

$$v(\xi) = \gamma_{\text{ref}}\,\hat{v}(\xi). \tag{7.13}$$

Po podstawieniu do równania otrzymujemy:

$$\gamma_{\text{ref}}\frac{d^4 \hat{v}(\xi)}{d\xi^4} + \frac{3p\,l^4}{2E\,b\,c^3} = 0. \tag{7.14}$$

Współczynnik γ_{ref} wybieramy tak, aby równanie przyjęło prostą postać bezwymiarową:

$$\frac{d^4 \hat{v}(\xi)}{d\xi^4} + 1 = 0, \qquad \gamma_{\text{ref}} = \frac{3p\,l^4}{2E\,b\,c^3}. \tag{7.15}$$

Wprowadzenie γ_{ref} jest istotne, ponieważ normalizuje skalę rozwiązania i sprowadza wszystkie składniki funkcji kosztu do porównywalnego rzędu wielkości. Dzięki temu poprawia się stabilność i szybkość uczenia modelu PINN, ponieważ sieć nie musi jednocześnie reprezentować bardzo małych i bardzo dużych wartości w funkcji straty.

Uogólnione siły przekrojowe i składowe stanu naprężenia obliczamy wg poniższych wzorów:

$$M_z(\xi) = -EJ_z\,\frac{\gamma_{\text{ref}}}{l^2}\frac{d^2\hat{v}(\xi)}{d\xi^2}, \qquad T_y(\xi) = -EJ_z\,\frac{\gamma_{\text{ref}}}{l^3}\frac{d^3\hat{v}(\xi)}{d\xi^3}. \tag{7.16}$$

$$\sigma_{xx} = \frac{M_z\,y}{J_z}, \quad \sigma_{xy} = \frac{T_y\,S_z}{b\,J_z}, \quad \text{gdzie} \quad S_z = b\,(c-y)\left(y + \frac{c-y}{2}\right) = \frac{b}{2}\left(c^2 - y^2\right). \tag{7.17}$$

Warunki brzegowe dla belki wspornikowej przyjmują postać:

$$v(1) = 0, \qquad v'(1) = 0, \qquad M_z(0) = 0, \qquad T_y(0) = 0. \tag{7.18}$$

Ostatecznie funkcję ugięcia i siły przekrojowe otrzymujemy w postaci:

$$v(\xi) = \frac{p\,l^4}{16\,E\,b\,c^3}\left(4\xi - \xi^4 - 3\right), \qquad M_z(\xi) = \frac{1}{2}\,p\,l^2\,\xi^2, \qquad T_y(\xi) = p\,l\,\xi. \tag{7.19}$$

Wprowadzając drugą współrzędną bezwymiarową $\eta = y/2c = y/\lambda l$, gdzie $\lambda = 2c/l$ jest bezwymiarowym parametrem geometrycznym opisującym proporcję wysokości do długości belki, składowe stanu naprężenia przyjmują postać:

$$\begin{aligned} \sigma_{xx}(\xi,\eta) &= \frac{3p}{4\,b\,c^3}\,(\xi l)^2\,\eta 2c = \frac{6p}{b\lambda^2}\xi^2\eta, \\ \sigma_{xy}(\xi,\eta) &= \frac{3p}{4\,b\,c^3}\,\xi l\,\left[c^2-(\eta 2c)^2\right] = \frac{3p}{2\,b\,\lambda}\,\xi\,\left(1-4\eta^2\right). \end{aligned} \tag{7.20}$$

Wyrażenia te, zapisane w zmiennych wymiarowych, będą potrzebne w dalszej części analizy.

7.3.2. Rozwiązanie przy zastosowaniu sieci PINN wg teorii belek

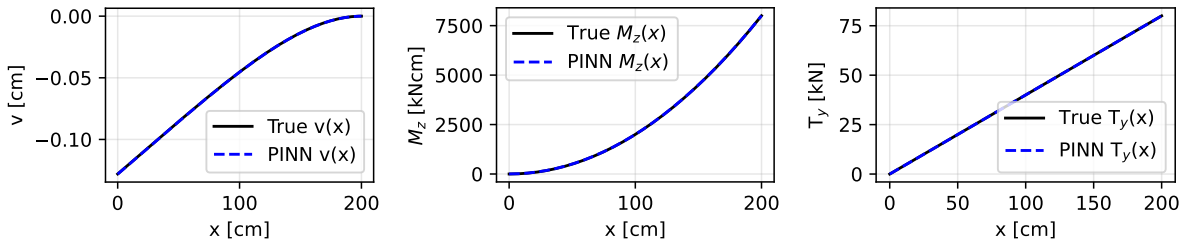
W rozwiązaniu przy zastosowaniu metody PINN rozpatrzono trzy warianty jednowymiarowego problemu zginania belki wspornikowej:

- w sformułowaniu wymiarowym,
- w sformułowaniu bezwymiarowym z równaniem różniczkowym odpowiadającym zależności (7.12),
- w sformułowaniu bezwymiarowym z równaniem różniczkowym odpowiadającym zależności (7.15).

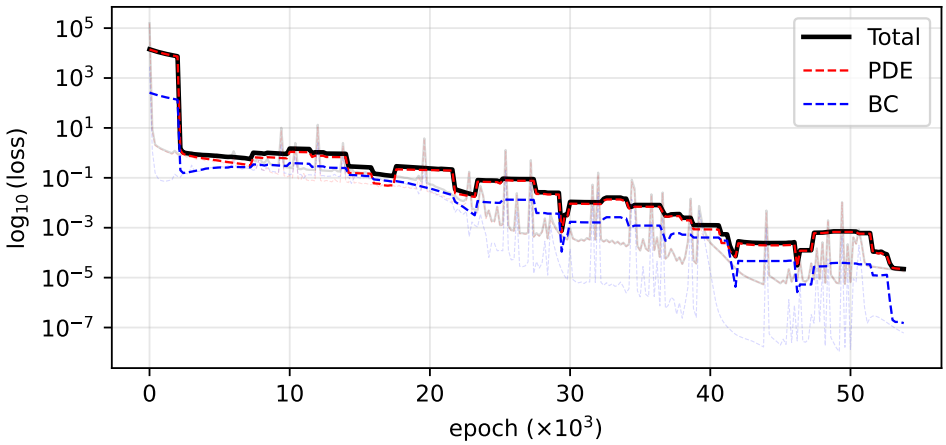
Warunki brzegowe zadano zgodnie z równaniem (7.18).

Do uczenia zastosowano wielowarstwową sieć neuronową o architekturze 1–50–50–50–1 z funkcją aktywacji typu `tanh` oraz inicjalizacją wag metodą `Glorot uniform` [38]. W obszarze zadania belki ustalono 128 punktów residuum PDE, natomiast warunki brzegowe narzucono w dwóch punktach na każdym końcu przedziału. Funkcja straty zbudowana została jako suma ważona reszty równania różniczkowego (PDE) oraz czterech składowych odpowiadających warunkom brzegowym, przyjmując jednostkowe wagi wszystkich składników. Dodatkowo użyto 100 punktów walidacyjnych w celu monitorowania błędu. Proces uczenia prowadzono z użyciem optymalizatora Adam z krokiem uczenia 10^{-3}, stosując mechanizm wczesnego zatrzymania na podstawie błędu funkcji kosztu obliczanej na zbiorze walidacyjnym. Przebieg funkcji straty w trakcie uczenia przedstawiono na rysunku 7.4d. Jako rozwiązanie referencyjne wykorzystano analityczną postać ugięcia belki, co umożliwiło bieżącą ocenę jakości aproksymacji w punktach walidacyjnych.

Wpływ normalizacji zmiennych na proces uczenia oceniono na podstawie błędu średniokwadratowego, obliczanego na zbiorze 100 punktów walidacyjnych równomiernie rozłożonych w dziedzinie obliczeniowej dla trzech wariantów zadania. Dla sformułowania



(a) Ugięcie $\mathsf{v}(x)$ (b) Moment $\mathsf{M_z}(x)$ (c) Siła tnąca $\mathsf{T_y}(x)$

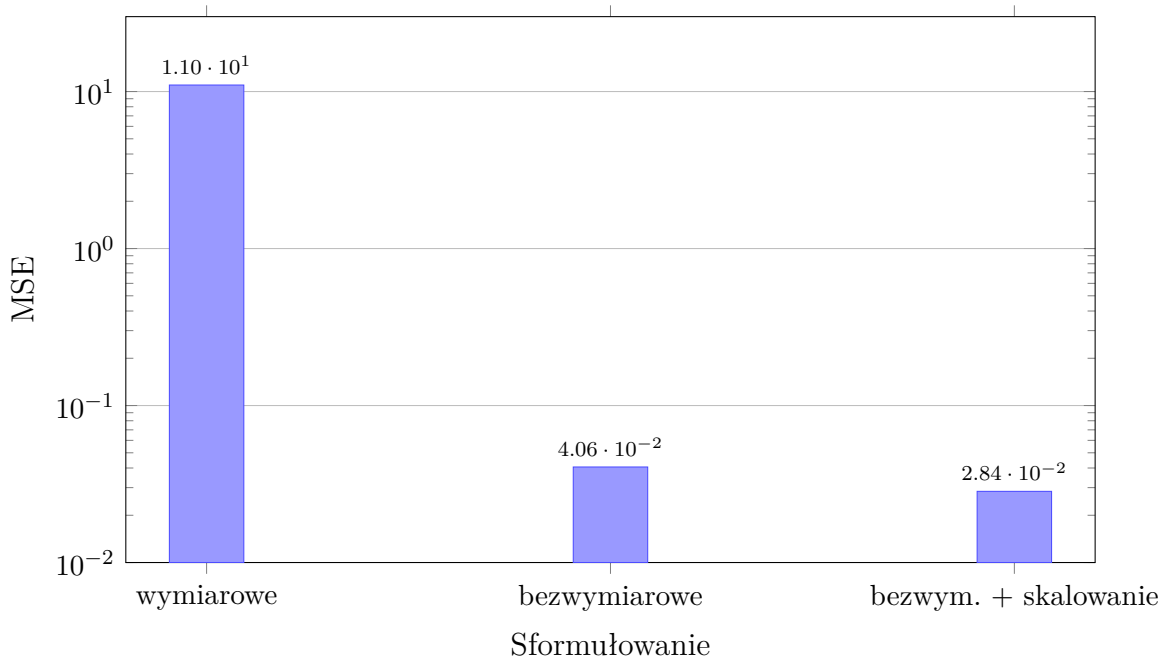


(d) Zależność funkcji straty od liczby epok uczenia na zbiorze treningowym: linia ciągła przedstawia stratę całkowitą, czerwona linia kropkowana odpowiada składnikowi PDE, natomiast niebieska linia przerywana — składnikom warunków brzegowych. Krzywe wygładzone zaprezentowano w postaci średniej kroczącej, natomiast półprzezroczyste przebiegi przedstawiają wartości surowe.

Rysunek 7.4. Porównanie wyników uzyskanych przy użyciu sieci PINN w sformułowaniu bezwymiarowym z równaniem różniczkowym odpowiadającym zależności (7.15) z rozwiązaniem analitycznym

problemu w zmiennych wymiarowych zaobserwowano brak zbieżności oraz wartość błędu MSE $= 1.10 \cdot 10^{1}$. Po wprowadzeniu współrzędnej bezwymiarowej zadanie zbiegło się i błąd zmniejszył się do MSE $= 4.06 \cdot 10^{-2}$, natomiast dodatkowe skalowanie ugięcia pozwoliło uzyskać dalszą poprawę do poziomu MSE $= 2.84 \cdot 10^{-2}$. Porównanie wartości błędu MSE dla kolejnych etapów normalizacji zestawiono na rysunku 7.5. Uzyskane wyniki potwierdzają znaczenie normalizacji zmiennych wejściowych w sieciach neuronowych oraz wskazują, że w przypadku sieci PINN może być to kluczowe dla uzyskania poprawnego rozwiązania. Wynika to z faktu, że typowe funkcje aktywacji wykazują najsilniejszą nieliniowość w ograniczonym zakresie argumentów (zwykle w pobliżu zera). Odpowiednie przeskalowanie zmiennych powoduje, że neurony pracują w tym „aktywnym" obszarze, co poprawia propagację gradientów i stabilność optymalizacji. W konsekwencji wprowa-

dzenie zmiennych bezwymiarowych oraz skalowanie wielkości fizycznych prowadzą do efektywniejszego uczenia i lepszej zbieżności modelu PINN. Zaobserwowane trudności ze zbieżnością ilustrują jedno z kluczowych ograniczeń metody PINN. W funkcji straty sieci współwystępują składniki o różnej naturze fizycznej — residuum równania różniczkowego, warunki brzegowe i ewentualnie warunki początkowe — których wartości mogą różnić się o wiele rzędów wielkości. Prowadzi to do silnie niezbalansowanego problemu optymalizacji, w którym gradienty jednych składników dominują nad pozostałymi, utrudniając jednoczesne spełnienie wszystkich warunków. Wang, Teng i Perdikaris [136] jako jedni z pierwszych szczegółowo zidentyfikowali tę patologię przepływu gradientów w sieciach PINN i zaproponowali algorytm Learning Rate Annealing, w którym wagi poszczególnych składników straty są adaptacyjnie dostosowywane na podstawie statystyk propagowanych gradientów Bischof i Kraus [34] rozwinęli to podejście, proponując schemat ReLoBRaLo (Relative Loss Balancing with Random Lookback), który dynamicznie równoważy wkłady poszczególnych członów straty i wykazuje wyższą dokładność przy mniejszym narzucie obliczeniowym w porównaniu z wcześniejszymi metodami. Stosowanie zmiennych bezwymiarowych, przedstawione w niniejszej pracy pozwala ograniczyć ten problem u źródła — sprowadzenie wielkości do porównywalnych zakresów zmniejsza dysproporcję między składnikami straty jeszcze przed rozpoczęciem uczenia.



Rysunek 7.5. Wartość błędu MSE dla kolejnych etapów normalizacji wejść: sformułowanie wymiarowe (MSE $= 1.10 \cdot 10^{1}$), wprowadzenie współrzędnej bezwymiarowej (MSE $= 4.06 \cdot 10^{-2}$) oraz dodatkowe skalowanie ugięcia (MSE $= 2.84 \cdot 10^{-2}$). Oś pionową wykonano w skali logarytmicznej.

7.4. Tarcza wspornikowa obciążona równomiernie

7.4.1. Rozwiązanie płaskiego zadania PSN wg teorii sprężystości metodą półodwrotną

W niniejszej części rozważono to samo zadanie wspornika obciążonego równomiernie na całej długości, rozwiązane jednak jako dwuwymiarowe zagadnienie płaskiego stanu naprężenia (PSN) w ramach teorii sprężystości. Model ten stanowi punkt odniesienia do porównania z elementarną teorią belek oraz do weryfikacji implementacji numerycznych PINN. Rozwiązanie analityczne przyjęto na podstawie rękopisu do Wykładu z przedmiotu Teorii Sprężystości i Plastyczności [135].

Naprężeniowe warunki brzegowe w rozpatrywanej tarczy można zapisać w postaci

$$\begin{aligned} \sigma_{xx} n_x + \sigma_{xy} n_y &= p_x, \\ \sigma_{yx} n_x + \sigma_{yy} n_y &= p_y. \end{aligned} \tag{7.21}$$

gdzie n_x oraz n_y są składowymi wektora normalnego do brzegu, natomiast p_x i p_y oznaczają składowe wektora obciążenia powierzchniowego. W przyjętym układzie mamy następujące

naprężeniowe warunki brzegowe:

$$y = c,\ x \in (0;l): \quad \begin{cases} n_x = 0 \\ n_y = 1 \end{cases} \wedge \begin{cases} p_x = 0 \\ p_y = -q \end{cases} \rightarrow \begin{cases} \sigma_{xy} = 0 \\ \sigma_{yy} = -q \end{cases}, \tag{7.22}$$

$$y = -c,\ x \in (0;l): \quad \begin{cases} n_x = 0 \\ n_y = -1 \end{cases} \wedge \begin{cases} p_x = 0 \\ p_y = 0 \end{cases} \rightarrow \begin{cases} \sigma_{xy} = 0 \\ \sigma_{yy} = 0 \end{cases}, \tag{7.23}$$

$$x = 0,\ y \in (-c;c): \quad \begin{cases} n_x = -1 \\ n_y = 0 \end{cases} \wedge \begin{cases} p_x = 0 \\ p_y = 0 \end{cases} \rightarrow \begin{cases} \sigma_{xx} = 0 \\ \sigma_{xy} = 0 \end{cases}. \tag{7.24}$$

W przypadku belek wysokich metoda półodwrotna polega na przyjęciu rozwiązania wynikającego z teorii belek oraz jego odpowiedniej modyfikacji w taki sposób, aby spełnione były równania zagadnienia brzegowego tarczy PSN. Z rozwiązania otrzymanego w ramach teorii belek (por. $(7.20)_1$) mamy:

$$\sigma_{xx} = \frac{3q}{4c^3} x^2 y. \tag{7.25}$$

Przyjmując powyższą postać naprężenia, dążymy do spełnienia równań równowagi wewnętrznej:

$$\begin{aligned} \frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y} &= 0, \\ \frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} &= 0. \end{aligned} \tag{7.26}$$

Z pierwszego równania równowagi (7.26) otrzymuje się:

$$\sigma_{xy} = -\int \frac{\partial \sigma_{xx}}{\partial x}\, dy + f_1(x) = -\frac{3q}{4c^3} x y^2 + f_1(x). \tag{7.27}$$

Nieznaną funkcję $f_1(x)$ wyznacza się z warunków brzegowych $(7.22)_1$ oraz $(7.23)_1$:

$$-\frac{3q}{4c^3} x(\pm c)^2 + f_1(x) = 0 \quad \Rightarrow \quad f_1(x) = \frac{3q}{4c} x. \tag{7.28}$$

Ostatecznie otrzymujemy składową styczną tensora naprężeń w postaci:

$$\sigma_{xy} = \frac{3q}{4c^3} x \left(c^2 - y^2\right). \tag{7.29}$$

Zauważmy, że otrzymaliśmy identyczne rozwiązanie jak w przypadku rozwiązania z teorii belek (por. $(7.20)_2$). Z drugiego równania równowagi (7.26) otrzymuje się:

$$\sigma_{yy} = -\int \frac{\partial \sigma_{xy}}{\partial x}\, dy + f_2(x) = -\frac{3q}{4c^3}\int (c^2 - y^2)\, dy + f_2(x) = -\frac{3q}{4c^3}\left(c^2 y - \frac{1}{3}y^3\right) + f_2(x) \tag{7.30}$$

Nieznaną funkcję $f_2(x)$ wyznaczamy z warunku brzegowego $(7.22)_2$, tj.

$$-\frac{3q}{4c^3}\left(c^3 - \frac{1}{3}c^3\right) + f_2(x) = -q \quad \Rightarrow \quad f_2(x) = -\frac{q}{2}. \tag{7.31}$$

Zauważmy, że warunek brzegowy $(7.23)_2$ jest także spełniony. Otrzymano:

$$\sigma_{yy} = \frac{q}{4c^3} y\left(y^2 - 3c^2\right) - \frac{q}{2}. \tag{7.32}$$

Otrzymano w ten sposób rozwiązanie w naprężeniach, które spełnia równania równowagi. Jest to rozwiązanie jakościowo inne niż rozwiązanie (7.20), gdyż otrzymano niezerowe naprężenia σ_{yy}. W ramach teorii tarcz nie jest to jednak rozwiązanie w pełni poprawne, gdyż ślad tensora płaskiego stanu naprężenia musi spełniać równanie Laplace'a:

$$\nabla^2(\sigma_{xx} + \sigma_{yy}) = 0, \qquad \text{czyli} \qquad \frac{\partial^2 \sigma_{xx}}{\partial x^2} + \frac{\partial^2 \sigma_{xx}}{\partial y^2} + \frac{\partial^2 \sigma_{yy}}{\partial x^2} + \frac{\partial^2 \sigma_{yy}}{\partial y^2} = 0. \tag{7.33}$$

które wynika z warunku nierozdzielności odkształceń i równań równowagi wyrażonych przez naprężenia. Obliczając odpowiednie pochodne składowych stanu naprężenia otrzymujemy:

$$\nabla^2\left(\sigma_{xx} + \sigma_{yy}\right) = \frac{3q}{c^3} y \neq 0, \tag{7.34}$$

co oznacza, iż warunek zgodności nie jest spełniony w całym obszarze rozważanej tarczy. W celu jego zapewnienia wprowadzono korektę składowej σ_{xx} w postaci:

$$\sigma_{xx} = \frac{3q}{4c^3} x^2 y + f(y), \tag{7.35}$$

gdzie $f(y)$ jest funkcją do wyznaczenia. Zauważono, że przyjęta modyfikacja nie wpływa na postacie składowych σ_{xy} oraz σ_{yy}, co wynika bezpośrednio z równań równowagi. Natomiast

warunek zgodności przyjmuje obecnie postać:

$$\frac{3q}{c^3}y + \frac{d^2f(y)}{dy^2} = 0. \tag{7.36}$$

Po dwukrotnym całkowaniu powyższego równania otrzymujemy:

$$f(y) = -\frac{q}{2c^3}y^3 + Ay + B, \tag{7.37}$$

gdzie A oraz B są stałymi całkowania. W konsekwencji składowa naprężenia σ_{xx} przyjmuje postać

$$\sigma_{xx} = \frac{3q}{4c^3}x^2y - \frac{q}{2c^3}y^3 + Ay + B, \tag{7.38}$$

co zapewnia spełnienie warunku zgodności. Warunki brzegowe na krawędzi $x = 0$ nie mogą zostać spełnione w sensie punktowym dla dowolnego $y \in (-c, c)$. W związku z tym zastosowano zasadę Saint-Venanta i narzucono osłabione (całkowe) warunki brzegowe w postaci:

$$x = 0: \qquad N_x = \int_{-c}^{c} \sigma_{xx}\, dy = 0, \qquad M_z = \int_{-c}^{c} y\, \sigma_{xx}\, dy = 0. \tag{7.39}$$

Z pierwszego warunku (7.39) otrzymano:

$$\int_{-c}^{c} \left(-\frac{q}{2c^3}y^3 + Ay + B\right) dy = 2Bc = 0 \quad \rightarrow \quad B = 0, \tag{7.40}$$

a z drugiego:

$$\int_{-c}^{c} \left(-\frac{q}{2c^3}y^4 + Ay^2\right) dy = -\frac{q}{5}c^2 + \frac{2A}{3}c^3 = 0, \tag{7.41}$$

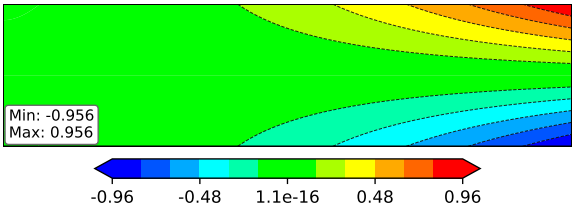
skąd wynika

$$A = \frac{3q}{10c}. \tag{7.42}$$

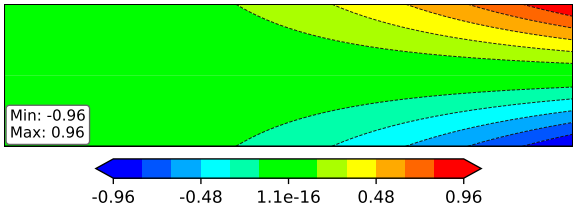
Ostatecznie pole stanu naprężenia opisane jest następującymi zależnościami:

$$\sigma_{xx} = \frac{q}{20c^3}y\left(15x^2 - 10y^2 + 6c^2\right),\ \sigma_{xy} = \frac{3q}{4c^3}x\left(c^2 - y^2\right),\ \sigma_{yy} = \frac{q}{4c^3}\left(y^3 - 3c^2y - 2c^3\right). \tag{7.43}$$

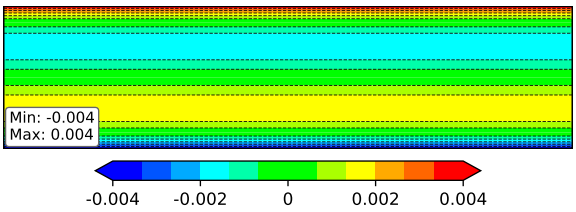
Stan odkształcenia wyznaczono na podstawie równań konstytutywnych dla płaskiego stanu naprężenia, podstawiając uprzednio otrzymane składowe tensora naprężeń (7.43).



(a) $\sigma_{xx}^{\mathrm{TS}}$ [kN/cm^2]

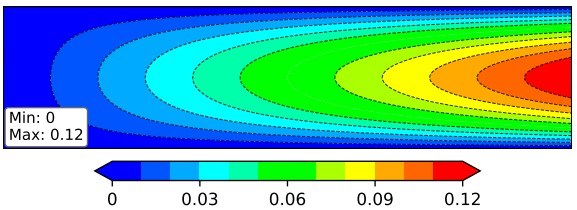


(b) $\sigma_{xx}^{\mathrm{WM}}$ [kN/cm^2]

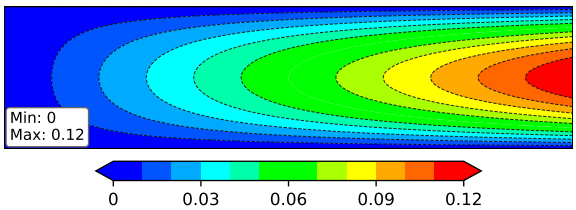


(c) Błąd $\Delta\sigma_{xx}$ [kN/cm^2]

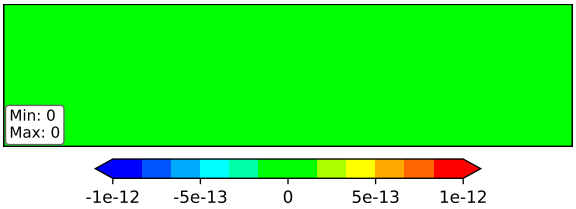
Rysunek 7.6. Wykresy warstwicowe składowej naprężenia σ_{xx} [kN/cm^2]: a) rozwiązanie z teorii sprężystości (TS), b) rozwiązanie z wytrzymałości materiałów (WM), c) błąd zdefiniowany jako $\Delta\sigma_{xx} = \sigma_{xx}^{\mathrm{TS}} - \sigma_{xx}^{\mathrm{WM}}$



(a) $\sigma_{xy}^{\mathrm{TS}}$ [kN/cm^2]



(b) $\sigma_{xy}^{\mathrm{WM}}$ [kN/cm^2]

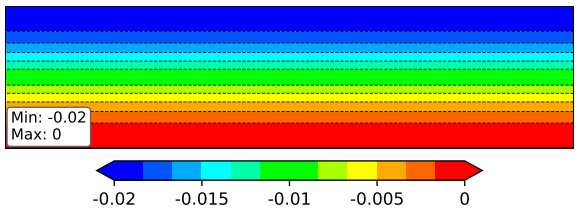


(c) Błąd $\Delta\sigma_{xy}$ [kN/cm^2]

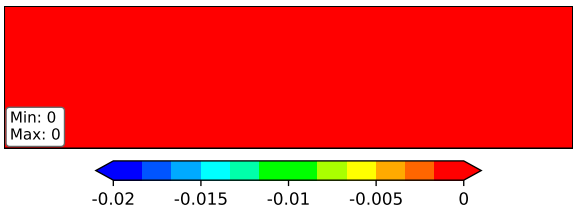
Rysunek 7.7. Wykresy warstwicowe składowej naprężenia σ_{xy} [kN/cm^2]: a) rozwiązanie z teorii sprężystości (TS), b) rozwiązanie z wytrzymałości materiałów (WM), c) błąd zdefiniowany jako $\Delta\sigma_{xy} = \sigma_{xy}^{\mathrm{TS}} - \sigma_{xy}^{\mathrm{WM}}$

Otrzymano następujące wzory na składowe tensora odkształcenia:

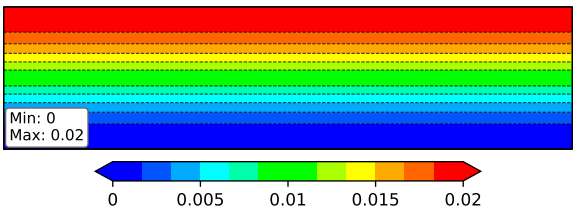
$$\begin{aligned}
\varepsilon_{xx} &= \frac{1}{E}\left(\sigma_{xx} - \nu\sigma_{yy}\right) = \frac{q}{20Ec^3}\Big[15x^2y - 5(2+\nu)y^3 + 3(2+5\nu)c^2y + 10\nu c^3\Big], \\
\varepsilon_{yy} &= \frac{1}{E}\left(\sigma_{yy} - \nu\sigma_{xx}\right) = \frac{q}{20Ec^3}\Big[5(1+2\nu)y^3 - 15\nu x^2y - 3(5+2\nu)c^2y - 10c^3\Big], \\
\gamma_{xy} &= \frac{2(1+\nu)}{E}\sigma_{xy} = \frac{3q(1+\nu)}{2Ec^3}x\left(c^2 - y^2\right)
\end{aligned} \tag{7.44}$$



(a) $\sigma_{yy}^{\mathrm{TS}}$ [kN/cm^2]



(b) $\sigma_{yy}^{\mathrm{WM}}$ [kN/cm^2]



(c) Błąd $\Delta\sigma_{yy}$ [kN/cm^2]

Rysunek 7.8. Wykresy warstwicowe składowej naprężenia σ_{yy} [kN/cm^2]: a) rozwiązanie z teorii sprężystości (TS), b) rozwiązanie z wytrzymałości materiałów (WM), c) błąd zdefiniowany jako $\Delta\sigma_{yy} = \sigma_{yy}^{\mathrm{TS}} - \sigma_{yy}^{\mathrm{WM}}$

oraz:

$$\varepsilon_{zz} = -\frac{\nu}{E}\left(\sigma_{xx} + \sigma_{yy}\right) = \frac{q\nu}{20Ec^3}\left(5y^3 - 15x^2y + 9c^2y + 10c^3\right). \tag{7.45}$$

Stan przemieszczeń wyznaczono poprzez całkowanie równań geometrycznych Cauchy'ego:

$$\varepsilon_{xx} = \frac{\partial u}{\partial x}, \qquad \varepsilon_{yy} = \frac{\partial v}{\partial y}, \qquad \gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}. \tag{7.46}$$

Całkując pierwsze z powyższych równań otrzymano:

$$u = \int \varepsilon_{xx}\, dx + f_3(y) = \frac{q}{20Ec^3}\left[5x^3y - 5(2+\nu)xy^3 + 3(2+5\nu)c^2xy + 10\nu c^3x\right] + f_3(y), \tag{7.47}$$

natomiast z drugiego równania uzyskano:

$$v = \int \varepsilon_{yy}\, dy + f_4(x) = \frac{q}{80Ec^3}\left[5(1+2\nu)y^4 - 30\nu x^2y^2 - 6(5+2\nu)c^2y^2 - 40c^3y\right] + f_4(x). \tag{7.48}$$

Podstawiając powyższe wyrażenia do trzeciego równania geometrycznego i uporządkowaniu wyrazów otrzymuje się równanie różniczkowe cząstkowe o zmiennych rozdzielonych:

$$\left\{\frac{df_4(x)}{dx} + \frac{q}{20Ec^3}\left(5x^3 - 24c^2x - 15\nu c^2x\right)\right\} + \left\{\frac{df_3(y)}{dy}\right\} = 0. \tag{7.49}$$

Ponieważ powyższe równanie musi być spełnione w każdym punkcie rozważanego obszaru, wyrażenia ujęte w nawiasach klamrowych muszą być stałe. Prowadzi to do następującego układu równań różniczkowych zwyczajnych na poszukiwane funkcje całkowania:

$$\frac{df_3(y)}{dy} = -D, \qquad \frac{df_4(x)}{dx} + \frac{q}{20Ec^3}\Big(5x^3 - 24c^2x - 15\nu c^2 x\Big) = D, \tag{7.50}$$

gdzie D jest stałą. Całkując otrzymano:

$$f_3(y) = -Dy + F, \quad f_4(x) = Dx + H - \frac{q}{80Ec^3}\Big(5x^4 - 48c^2x^2 - 30\nu c^2 x^2\Big), \tag{7.51}$$

gdzie F i H są stałymi całkowania. Stałe D, F oraz H wyznaczane są z przemieszczeniowych warunków brzegowych:

$$x = l, \quad y \in (-c, c): \qquad u = 0, \quad v = 0. \tag{7.52}$$

Po podstawieniu otrzymanych wyrażeń do wzorów na przemieszczenia:

$$u = \frac{q}{20Ec^3}\Big[5x^3y - 5(2+\nu)xy^3 + 3(2+5\nu)c^2xy + 10\nu c^3 x\Big] - Dy + F, \tag{7.53}$$

$$v = \frac{q}{80Ec^3}\Big[5(1+2\nu)y^4 + 30\nu x^2(c^2 - y^2) - 6(5+2\nu)c^2y^2 - 40c^3y + 48c^2x^2 - 5x^4\Big] \tag{7.54}$$

$$+ Dx + H$$

okazuje się, że warunki te nie mogą zostać spełnione jednocześnie dla żadnych wartości stałych D, F oraz H. W związku z tym, analogicznie jak w przypadku warunków naprężeniowych, zastosowano osłabione warunki kinematyczne określone w punkcie podparcia. Zamiast (7.52) przyjmujemy warunki brzegowe o postaci:

$$x = l, \quad y = 0: \qquad u = 0, \quad v = 0, \quad \frac{\partial u}{\partial y} = 0. \tag{7.55}$$

przy czym zamiast ostatniego z powyższych warunków można alternatywnie rozważać warunek:

$$\frac{\partial v}{\partial x} = 0. \tag{7.56}$$

Stosując $(7.55)_1$ otrzymano:

$$\frac{q}{20Ec^3}\left(10\nu c^3 l\right) + F = 0, \qquad \Rightarrow \qquad F = -\frac{q\nu l}{2E}, \tag{7.57}$$

natomiast z $(7.55)_2$ wynika:

$$\frac{q}{80Ec^3}\left(30\nu c^2 l^2 + 48c^2 l^2 - 5l^4\right) + Dl + H = 0. \tag{7.58}$$

Wyznaczając pochodną funkcji przemieszczenia poziomego względem zmiennej y otrzymano:

$$\frac{\partial u}{\partial y} = \frac{q}{20Ec^3}\Big[5x^3 - 15(2+\nu)xy^2 + 3(2+5\nu)c^2 x\Big] - D. \tag{7.59}$$

Podstawiając powyższe wyrażenie do trzeciego z osłabionych kinematycznych warunków brzegowych w punkcie podparcia $(7.55)_3$ otrzymano:

$$\left.\frac{\partial u}{\partial y}\right|_{x=l,y=0} = \frac{q}{20Ec^3}\Big[5l^3 + 3(2+5\nu)c^2 l\Big] - D = 0. \tag{7.60}$$

Stąd bezpośrednio wynikają wartości stałych:

$$D = \frac{q}{20Ec^3}\Big[5l^3 + 3(2+5\nu)c^2 l\Big], \qquad H = -\frac{q}{80Ec^3}\Big[90\nu c^2 l^2 + 72c^2 l^2 + 15l^4\Big]. \tag{7.61}$$

Ostatecznie pole przemieszczeń przyjmuje postać:

$$u(x,y) = \frac{q}{20Ec^3}\Big\{5(x^3 - l^3)y - 5(2+\nu)xy^3 + 3(2+5\nu)c^2(x-l)y + 10\nu c^3(x-l)\Big\}, \tag{7.62}$$

$$v(x,y) = -\frac{q}{80Ec^3}\Big[5(1+2\nu)y^4 - 30\nu x^2 y^2 - 6(5+2\nu)c^2 y^2 - 40c^3 y + 30\nu c^2(x^2 - 3l^2) + 24c^2(2x^2 - 3l^2) - 5(x^4 + 3l^4) + 20l^3 x + 12(2+5\nu)c^2 lx\Big].$$

Z postaci funkcji $u(x,y)$ wynika, że przekroje belki ($x = \text{const}$) nie pozostają płaskie. Dla osi belki ($y = 0$) przemieszczenie poziome przyjmuje prostą postać:

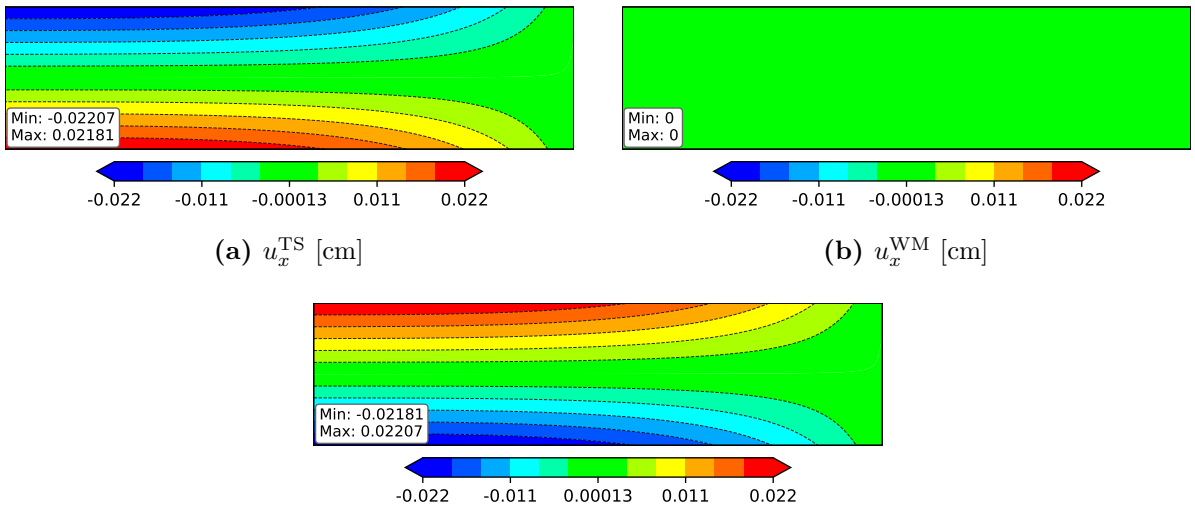
$$u(x) = \frac{q\nu}{2E}(x - l). \tag{7.63}$$

Analogicznie, pionowe przemieszczenie osi belki otrzymujemy jako:

$$v(x) = -\frac{q}{16Ec^3}\left(x^4 - 4l^3x + 3l^4\right) - \frac{3q}{40Ec}(l-x)\Big[3(4+5\nu)l + (8+5\nu)x\Big]. \tag{7.64}$$

Ugięcie końca belki wynosi zatem:

$$v_{\text{extr}}^{(\text{PSN})} = v(0) = -\frac{3ql^4}{16Ec^3}\left[1 + \frac{3(4+5\nu)}{10}\left(\frac{2c}{l}\right)^2\right]. \tag{7.65}$$



(a) u_x^{TS} [cm] **(b)** u_x^{WM} [cm]

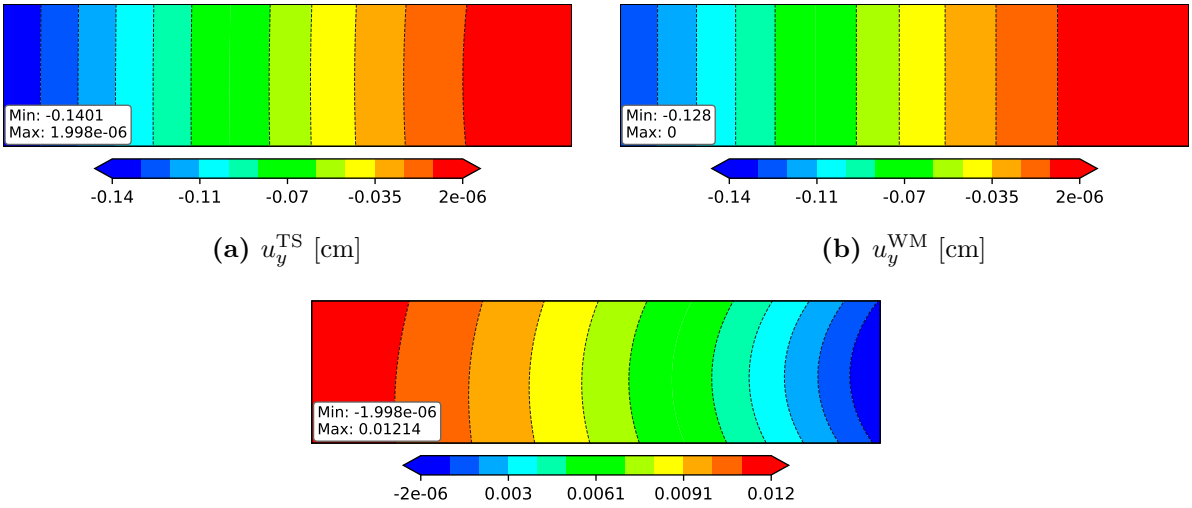
(c) Błąd Δu_x [cm]

Rysunek 7.9. Wykresy warstwicowe składowej ugięcia u_x [cm]: a) rozwiązanie uzyskane z teorii sprężystości (TS), b) rozwiązanie uzyskane z wytrzymałości materiałów (WM) oraz c) błąd zdefiniowany jako $\Delta(u_x) = u_x^{\text{TS}} - u_x^{\text{WM}}$

7.4.2. Rozwiązanie MES

W celu weryfikacji rozwiązań analitycznych zadanie rozwiązano metodą elementów skończonych (MES) w programie Abaqus [80]. Rozpatrywany model tarczowy belki wspornikowej zbudowano w płaskim stanie naprężenia (PSN), co jest spójne z przyjętymi wcześniej założeniami teorii sprężystości.

Zaproponowano regularną siatkę elementów skończonych złożoną z elementów prostokątnych o jednakowej długości krawędzi wynoszącej 0.5 [cm] w obu kierunkach. Łącznie model składa się z 40 000 elementów skończonych i 40 501 węzłów. Zastosowano czterowęzłowe elementy skończone typu `CPS4R`, tj. elementy płaskiego stanu naprężenia z liniowymi funkcjami kształtu i zredukowanym całkowaniem.



(a) u_y^{TS} [cm]

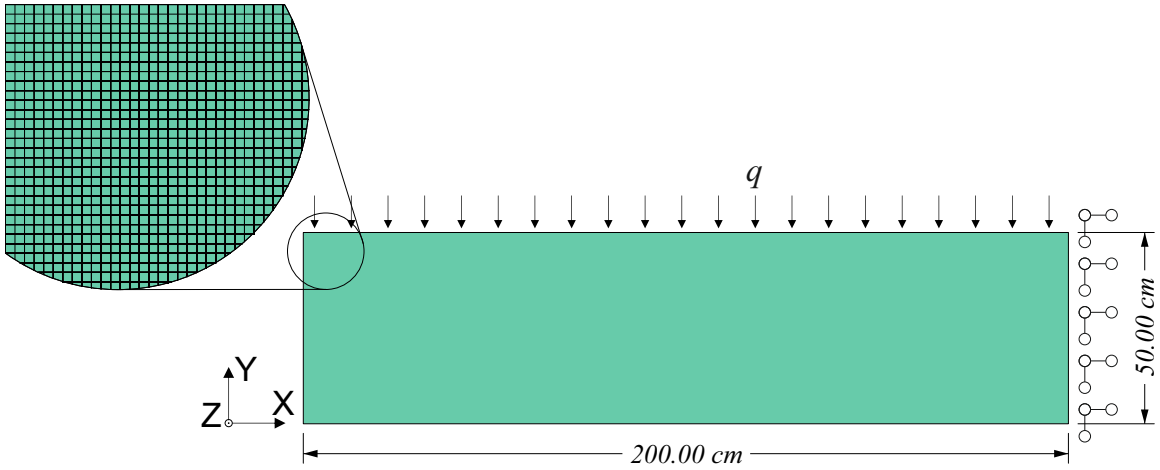
(b) u_y^{WM} [cm]

(c) Błąd Δu_y [cm]

Rysunek 7.10. Wykresy warstwicowe składowej ugięcia u_y [cm]: a) rozwiązanie uzyskane z teorii sprężystości (TS), b) rozwiązanie uzyskane z wytrzymałości materiałów (WM) oraz c) błąd zdefiniowany jako $\Delta(u_y) = u_y^{\mathrm{TS}} - u_y^{\mathrm{WM}}$

Materiał modelowany jest jako liniowo sprężysty i izotropowy. Moduł Younga E oraz współczynnik Poissona ν przyjęto na podstawie tabeli 7.1.

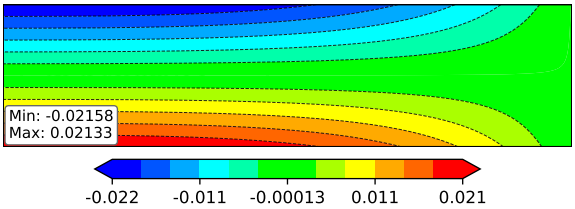
Na prawym brzegu tarczy zablokowano przemieszczenia we wszystkich węzłach leżących na tej krawędzi. Obciążenie zrealizowano jako ciśnienie powierzchniowe (opcja `*Dsload`) o wartości q przyłożone do górnej krawędzi tarczy, co odpowiada równomiernemu obciążeniu liniowemu $p = q \cdot b$.



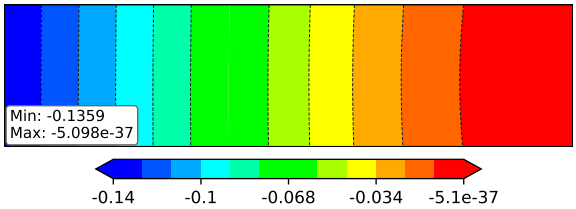
Rysunek 7.11. Model obliczeniowy z warunkami brzegowymi i siatką elementów skończonych

Liniowe zadanie statyki, przy założeniu małych przemieszczeń i odkształceń, rozwiązano w jednym kroku obliczeniowym. Na rys. 7.12 przedstawiono wykresy warstwicowe

przemieszczeń poziomych u_x oraz pionowych u_y, natomiast na rys. 7.13 zamieszczono rozkłady składowych σ_{xx}, σ_{yy}, σ_{xy} tensora naprężeń.

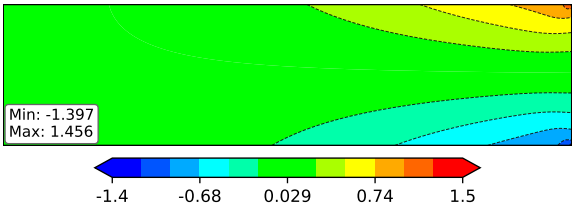


(a) u_x [cm]

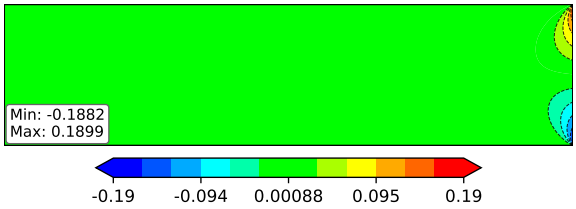


(b) u_y [cm]

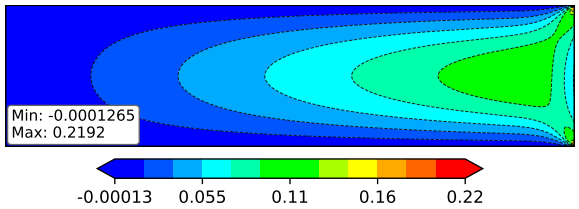
Rysunek 7.12. Wykresy warstwicowe przemieszczeń: a) składowa pozioma u_x, b) składowa pionowa u_y



(a) σ_{xx} [kN/cm^2]



(b) σ_{yy} [kN/cm^2]



(c) σ_{xy} [kN/cm^2]

Rysunek 7.13. Wykresy warstwicowe składowych tensora naprężeń: a) σ_{xx}, b) σ_{yy}, c) σ_{xy}

7.4.3. Rozwiązanie zadania PSN przy pomocy sieci PINN

W celu zbadania wpływu sformułowania, architektury sieci oraz sposobu wymuszania warunków brzegowych na jakość aproksymacji, szybkość zbieżności i czas obliczeń, rozpatrzono cztery warianty rozwiązania zadania tarczy wspornikowej metodą PINN. Warianty różnią się liczbą wyjść sieci (2 lub 5), typem architektury (FNN/PFNN) oraz rodzajem warunków brzegowych (miękkie/twarde). Warianty oznaczono symbolami W1–W4, a ich charakterystykę zestawiono w tabeli 7.2.

Wariant W1 korzysta z sformułowania przemieszczeniowego — sieć aproksymuje wyłącznie pola przemieszczeń $(\hat{u}, \hat{v})$, naprężenia obliczane są z równań konstytutywnych. Warianty

Tabela 7.2. Zestawienie badanych wariantów PINN dla tarczy wspornikowej

	W1	**W2**	**W3**	**W4**
Sformułowanie	przemieszczeniowe	mieszane	mieszane	mieszane
Architektura	FNN	FNN	PFNN	FNN
Liczba wyjść	2	5	5	5
Wyjścia	$\hat{u}$, $\hat{v}$		$\hat{u}$, $\hat{v}$, $\hat{\sigma}_{xx}$, $\hat{\sigma}_{yy}$, $\hat{\sigma}_{xy}$	
Residua PDE	3	6	6	6
— równowagi	2	2	2	2
— zgodności	1	1	1	1
— konstytutywne	—	3	3	3
Warunki brzeg.	soft	soft	soft	hard
BC naprężeniowe	`OperatorBC`	`DirichletBC`	`DirichletBC`	transf. (7.84)

W2–W4 stosują sformułowanie mieszane z pięcioma wyjściami, w którym sieć aproksymuje jednocześnie przemieszczenia i naprężenia. Wariant W3 jako jedyny wykorzystuje architekturę PFNN, umożliwiającą niezależną parametryzację poszczególnych składowych. Wariant W4 stosuje twarde warunki brzegowe, realizowane przez transformację wyjścia sieci (7.84), dzięki czemu warunki brzegowe są spełnione ściśle. Wspólne dla wszystkich wariantów elementy — bezwymiarowość, skalowanie oraz przyjęte warunki brzegowe, a także szczegóły poszczególnych wariantów przedstawiono poniżej.

Bezwymiarowaść i skalowanie

Na podstawie analizy rozwiązania jednowymiarowego stwierdzono konieczność zastosowania podejścia bezwymiarowego. Wprowadzono bezwymiarowe współrzędne przestrzenne:

$$\xi = \frac{x}{l} \;\rightarrow\; x = \xi\, l, \qquad \eta = \frac{y}{2c} = \frac{y}{\lambda l} \;\rightarrow\; y = \lambda\, l\, \eta, \tag{7.66}$$

gdzie l oznacza długość tarczy, a $2c$ jej wysokość oraz bezwymiarowy parametr geometryczny:

$$\lambda = \frac{2c}{l} \;\rightarrow\; c = \frac{1}{2}\lambda\, l, \tag{7.67}$$

opisujący proporcję wysokości do długości tarczy. Operatory różniczkowe we współrzędnych bezwymiarowych przyjmują postać:

$$\frac{\partial}{\partial x} = \frac{1}{l}\frac{\partial}{\partial \xi}, \qquad \frac{\partial}{\partial y} = \frac{1}{2c}\frac{\partial}{\partial \eta}. \tag{7.68}$$

Po wprowadzeniu bezwymiarowych współrzędnych składowe pola przemieszczeń przyjmują postać:

$$u(\xi,\eta) = \frac{q\,l}{10\,E\,\lambda^2}\Big[20\,\eta(\xi^3-1)-20(2+\nu)\lambda^2\,\xi\,\eta^3+3(2+5\nu)\,\lambda^2(\xi-1)\,\eta+5\nu\lambda^2(\xi-1)\Big], \quad (7.69)$$

$$\begin{aligned} v(\xi,\eta) = -\frac{q\,l}{20\,E\,\lambda^3}\Big[&10(1+2\nu)\,\lambda^4\eta^2 - 60\nu\,\xi^2\eta^2 - 3(5+2\nu)\,\lambda^4\eta^4 - 10\lambda^4\eta + 15\nu\lambda^2(\xi^2-3) \\ &+ 12\lambda^2(2\xi^2-3) - 5(\xi^4+3) + 20\xi + 12(2+5\nu)\,\xi\Big], \end{aligned} \quad (7.70)$$

a składowe tensora naprężeń:

$$\begin{aligned} \sigma_{xx}(\xi,\eta) &= \frac{q}{5\,\lambda^2}\,\eta\left(30\xi^2 - 20\lambda^2\eta^2 + 3\lambda^2\right), \\ \sigma_{xy}(\xi,\eta) &= \frac{3q}{2\,\lambda}\,\xi\left(1-4\eta^2\right), \\ \sigma_{yy}(\xi,\eta) &= \frac{q}{2}\left(4\eta^3 - 3\eta - 1\right). \end{aligned} \quad (7.71)$$

Z powyższych wyrażeń wynika, że poszczególne składowe pól skalują się z różnymi potęgami parametru λ. W celu ujednolicenia rzędu wielkości wyjść sieci neuronowej wprowadzono skale normalizujące:

$$\sigma_0 = \frac{q}{\lambda^k}, \qquad u_0 = \frac{\sigma_0\,l}{E}, \quad (7.72)$$

gdzie k jest wykładnikiem skalowania (w obliczeniach przyjęto $k = 2$). Bezwymiarowe pola wyjściowe sieci zdefiniowano jako $\hat{u}_i = u_i/u_0$ oraz $\hat{\sigma}_{ij} = \sigma_{ij}/\sigma_0$.

Warunki brzegowe

W wariantach W1–W3 (miękkie warunki brzegowe) na prawym brzegu przyjęto osłabione warunki kinematyczne (7.55), aby zachować zgodność z rozwiązaniem referencyjnym (TS) i umożliwić bezpośrednie porównanie wyników.

Warunki naprężeniowe na górnej i dolnej krawędzi przyjęto we współrzędnych bezwymiarowych analogicznie do (7.22) i (7.23). Na lewej krawędzi ($\xi = 0$) w rozwiązaniu analitycznym zastosowano osłabione warunki brzegowe w postaci całkowej (7.39), wynikające z zasady Saint-Venanta, ponieważ przyjęta wielomianowa postać pola naprężeń nie pozwala na punktowe spełnienie warunku (7.24) dla dowolnego $y \in (-c,\, c)$. W implementacji PINN takie ograniczenie nie występuje — sieć neuronowa może kształtować

dowolny rozkład naprężeń na brzegu. Początkowo podjęto próbę zastosowania warunków osłabionych analogicznie do rozwiązania TS, jednak warunki te spełniane są przez sieć jedynie w przybliżeniu (jako składniki kary w funkcji kosztu), a powstałe residua okazały się wystarczające do wygenerowania niezerowego momentu na tej krawędzi. Zrezygnowano zatem z warunków osłabionych na rzecz silnych (punktowych) warunków brzegowych (7.24) na całej lewej krawędzi.

Ostatecznie warunki brzegowe we współrzędnych bezwymiarowych przyjmują postać:

$$\eta = \tfrac{1}{2},\ \xi \in (0;\,1): \qquad \hat{\sigma}_{xy} = 0, \quad \hat{\sigma}_{yy} = -\hat{q}, \tag{7.73}$$

$$\eta = -\tfrac{1}{2},\ \xi \in (0;\,1): \qquad \hat{\sigma}_{xy} = 0, \quad \hat{\sigma}_{yy} = 0, \tag{7.74}$$

$$\xi = 0,\ \eta \in (-\tfrac{1}{2};\,\tfrac{1}{2}): \qquad \hat{\sigma}_{xx} = 0, \quad \hat{\sigma}_{xy} = 0, \tag{7.75}$$

$$\xi = 1,\ \eta = 0: \qquad \hat{u} = 0, \quad \hat{v} = 0, \quad \frac{\partial \hat{u}}{\partial \eta} = 0. \tag{7.76}$$

gdzie $\hat{q} = q/\sigma_0$.

Procedura treningu

We wszystkich wariantach zastosowano jednolitą procedurę treningu. W bezwymiarowej domenie $[0,1] \times [-\frac{1}{2}, \frac{1}{2}]$ wygenerowano losowo 1000 punktów residuum w domenie (punkty PDE) oraz 500 punktów na brzegu, a dodatkowe 100 punktów testowych wykorzystano do monitorowania błędu aproksymacji względem rozwiązania analitycznego (TS). Funkcję kosztu zdefiniowano jako:

$$L = L_{\mathrm{PDE}} + L_{\mathrm{BC}}. \tag{7.77}$$

Trening przebiegał dwuetapowo. W pierwszej fazie zastosowano optymalizator Adam z krokiem uczenia 10^{-3} przez maksymalnie 10^6 iteracji, z mechanizmem wczesnego zatrzymania (*early stopping*) monitorującym wartość funkcji kosztu na zbiorze testowym (maksymalna liczba iteracji bez poprawy: $5 \cdot 10^4$). W drugiej fazie przeprowadzono dostrajanie metodą L-BFGS-B, co pozwoliło na dalszą redukcję residuów w otoczeniu minimum znalezionego przez Adama.

We wszystkich wariantach funkcją aktywacji była tanh, a wagi inicjalizowano metodą Glorot uniform [38].

Jako rozwiązanie referencyjne wykorzystano analityczną postać pól przemieszczeń i naprężeń (rozwiązanie TS), co umożliwiło bieżącą ocenę jakości aproksymacji poprzez obliczanie błędu względnego normyL^2 w punktach testowych. W wariancie W4 (twarde warunki brzegowe), w którym warunki brzegowe nie odpowiadają dokładnie rozwiązaniu TS, jako dodatkowe rozwiązanie referencyjne wykorzystano model MES. Kod źródłowy implementacji zamieszczono w załączniku 8.

Kod podzielono na moduł główny `train.py` oraz pakiet `src/` zawierający pięć modułów: `params.py` (parametry fizyczne, wielkości pochodne, operatory różniczkowe), `pde.py` (residua równań różniczkowych), `bcs.py` (warunki brzegowe miękkie i twarde), `solution.py` (rozwiązanie analityczne TS i MES) oraz `network.py` (budowa architektury FNN/PFNN). Wszystkie hiperparametry zebrane są w pliku `config.yaml`, co umożliwia uruchamianie poszczególnych wariantów bez modyfikacji kodu źródłowego.

Wariant W1 — FNN, sformułowanie w przemieszczeniach, miękkie warunki brzegowe

W sformułowaniu przemieszczeniowym sieć neuronowa aproksymuje pole przemieszczeń $\hat{\mathbf{u}} = (\hat{u},\, \hat{v})$, a naprężenia wyznaczane są pośrednio — z predykowanego pola przemieszczeń poprzez automatyczne różniczkowanie i związek konstytutywny (odwrotność zależności (7.44)):

$$\boldsymbol{\sigma} = \mathbf{C} \cdot \boldsymbol{\varepsilon}(\hat{\mathbf{u}}), \tag{7.78}$$

Funkcja kosztu L_{PDE} obejmuje trzy residua. Dwa odpowiadają bezwymiarowym równaniom równowagi:

$$R_{\mathrm{eq},x} = \frac{1}{\sigma_0}\left(\frac{1}{l}\,\frac{\partial \hat{\sigma}_{xx}}{\partial \xi} + \frac{1}{2c}\,\frac{\partial \hat{\sigma}_{xy}}{\partial \eta}\right), \qquad R_{\mathrm{eq},y} = \frac{1}{\sigma_0}\left(\frac{1}{l}\,\frac{\partial \hat{\sigma}_{xy}}{\partial \xi} + \frac{1}{2c}\,\frac{\partial \hat{\sigma}_{yy}}{\partial \eta}\right), \tag{7.79}$$

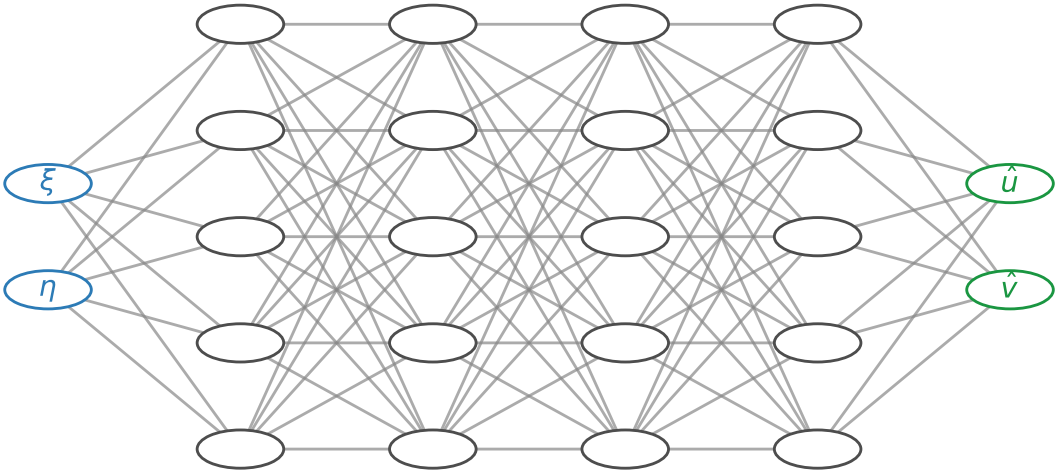
a trzecie — równaniu zgodności w naprężeniach:

$$R_{\mathrm{comp}} = \frac{1}{\sigma_0}\,\nabla^2(\sigma_{xx} + \sigma_{yy})\,. \tag{7.80}$$

Warunki brzegowe naprężeniowe na krawędziach tarczy realizowane są za pomocą operatora `OperatorBC`, który pozwala zdefiniować dowolny warunek w postaci $f(\hat{\mathbf{u}}) = 0$ na brzegu.

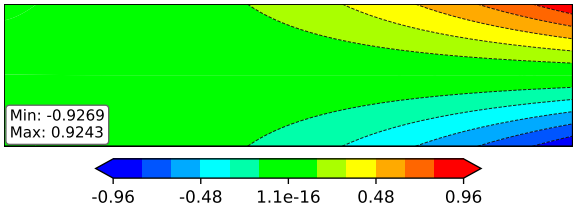
W tym przypadku operator wyznacza naprężenia z predykowanego pola przemieszczeń za pomocą zależności (7.78) i porównuje je z wartościami zadanymi.

Zastosowano architekturę FNN (sieć w pełni połączona) z czterema warstwami ukrytymi po 64 neurony i dwoma wyjściami. Schemat architektury przedstawiono na rys. 7.14.

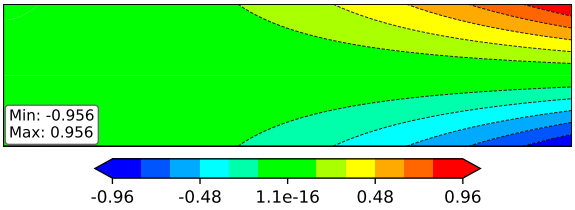


Rysunek 7.14. Schemat architektury sieci FNN w wariancie W1

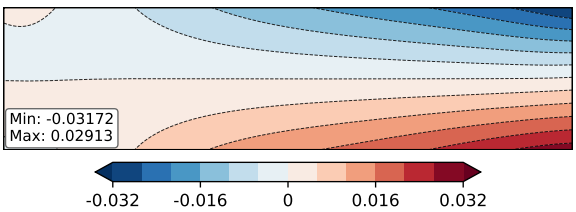
Porównanie przewidywanych pól przemieszczeń i naprężeń z rozwiązaniem referencyjnym (TS) przedstawiono na rys. 7.15–7.19.



(a) $\sigma_{xx}^{\mathrm{PINN^{W1}}}$ [kN/cm^2]

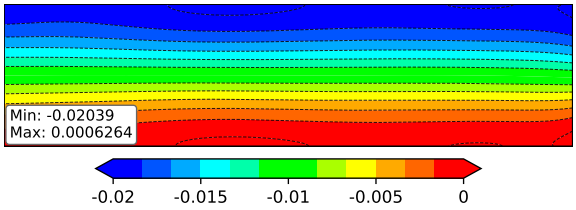


(b) $\sigma_{xx}^{\mathrm{TS}}$ [kN/cm^2]

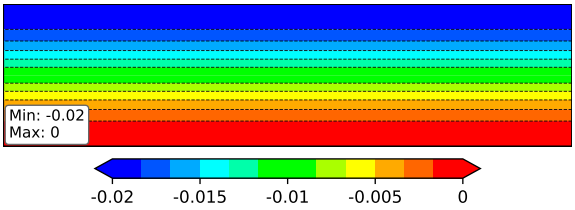


(c) Błąd $\Delta\sigma_{xx}$ [kN/cm^2]

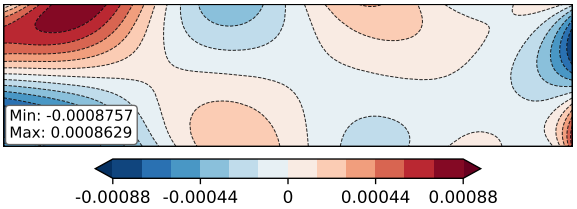
Rysunek 7.15. Wykresy warstwicowe składowej naprężenia σ_{xx} [kN/cm^2]: a) rozwiązanie z PINN w wariancie W1, b) rozwiązanie z teorii sprężystości (TS), c) błąd bezwzględny $\Delta\sigma_{xx} = \sigma_{xx}^{\mathrm{PINN^{W1}}} - \sigma_{xx}^{\mathrm{TS}}$



(a) $\sigma_{yy}^{\mathrm{PINN}^{\mathrm{W1}}}$ [kN/cm^2]

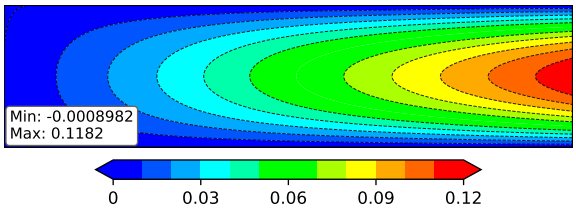


(b) $\sigma_{yy}^{\mathrm{TS}}$ [kN/cm^2]

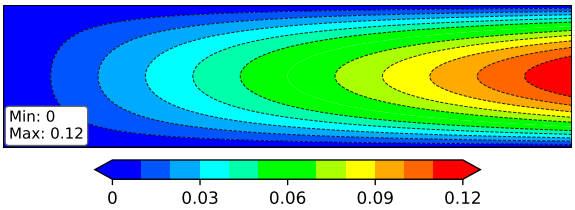


(c) Błąd $\Delta\sigma_{yy}$ [kN/cm^2]

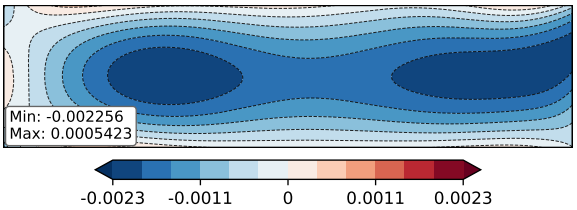
Rysunek 7.16. Wykresy warstwicowe składowej naprężenia σ_{yy} [kN/cm^2]: a) rozwiązanie z PINN w wariancie W1, b) rozwiązanie z teorii sprężystości (TS), c) błąd bezwzględny $\Delta\sigma_{yy} = \sigma_{yy}^{\mathrm{PINN}^{\mathrm{W1}}} - \sigma_{yy}^{\mathrm{TS}}$



(a) $\sigma_{xy}^{\mathrm{PINN}^{\mathrm{W1}}}$ [kN/cm^2]

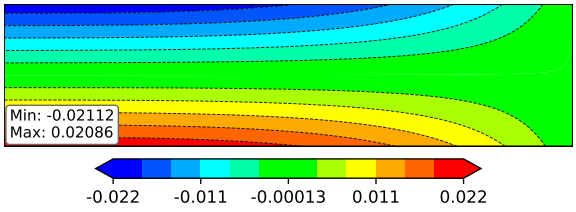


(b) $\sigma_{xy}^{\mathrm{TS}}$ [kN/cm^2]

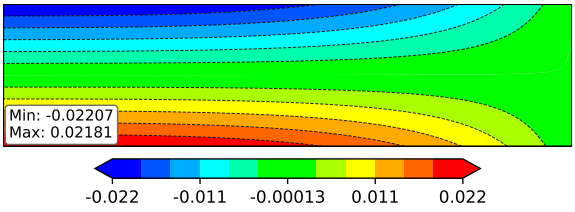


(c) Błąd $\Delta\sigma_{xy}$ [kN/cm^2]

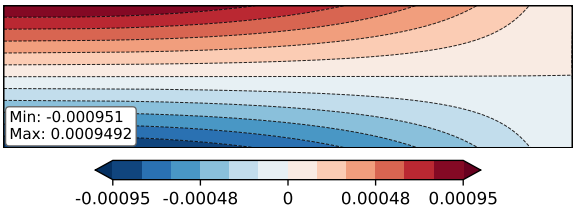
Rysunek 7.17. Wykresy warstwicowe składowej naprężenia σ_{xy} [kN/cm^2]: a) rozwiązanie z PINN w wariancie W1, b) rozwiązanie z teorii sprężystości (TS), c) błąd bezwzględny $\Delta\sigma_{xy} = \sigma_{xy}^{\mathrm{PINN}^{\mathrm{W1}}} - \sigma_{xy}^{\mathrm{TS}}$



(a) $u_x^{\mathrm{PINN}^{\mathrm{W1}}}$ [cm]

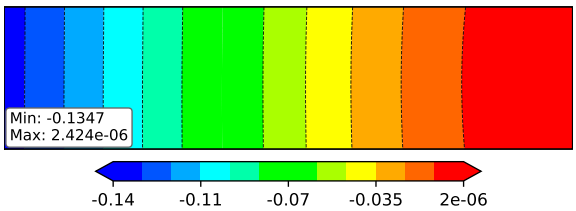


(b) u_x^{TS} [cm]

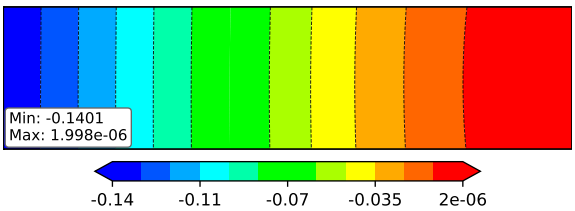


(c) Błąd Δu_x [cm]

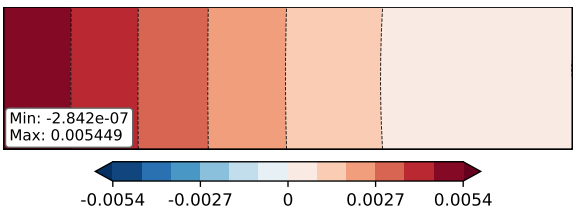
Rysunek 7.18. Wykresy warstwicowe przemieszczenia poziomego u_x [cm]: a) rozwiązanie z PINN w wariancie W1, b) rozwiązanie z teorii sprężystości (TS), c) błąd bezwzględny $\Delta u_x = u_x^{\mathrm{PINN}^{\mathrm{W1}}} - u_x^{\mathrm{TS}}$



(a) $u_y^{\mathrm{PINN}^{\mathrm{W1}}}$ [cm]



(b) u_y^{TS} [cm]



(c) Błąd Δu_y [cm]

Rysunek 7.19. Wykresy warstwicowe przemieszczenia pionowego u_y [cm]: a) rozwiązanie z PINN w wariancie W1, b) rozwiązanie z teorii sprężystości (TS), c) błąd bezwzględny $\Delta u_y = u_y^{\mathrm{PINN}^{\mathrm{W1}}} - u_y^{\mathrm{TS}}$

Wariant W2 — FNN, sformułowanie mieszane, miękkie warunki brzegowe

W sformułowaniu mieszanym sieć posiada pięć wyjść: dwa bezwymiarowe przemieszczenia ($\hat{u}$, $\hat{v}$) oraz trzy bezwymiarowe naprężenia ($\hat{\sigma}_{xx}$, $\hat{\sigma}_{yy}$, $\hat{\sigma}_{xy}$). Funkcja kosztu L_{PDE}

obejmuje sześć residuów. Dwa równania równowagi operują na wyjściach naprężeniowych sieci:

$$R_{\mathrm{eq},x} = \frac{1}{l}\frac{\partial \hat{\sigma}_{xx}}{\partial \xi} + \frac{1}{2c}\frac{\partial \hat{\sigma}_{xy}}{\partial \eta}, \qquad R_{\mathrm{eq},y} = \frac{1}{l}\frac{\partial \hat{\sigma}_{xy}}{\partial \xi} + \frac{1}{2c}\frac{\partial \hat{\sigma}_{yy}}{\partial \eta}, \tag{7.81}$$

równanie zgodności ma postać:

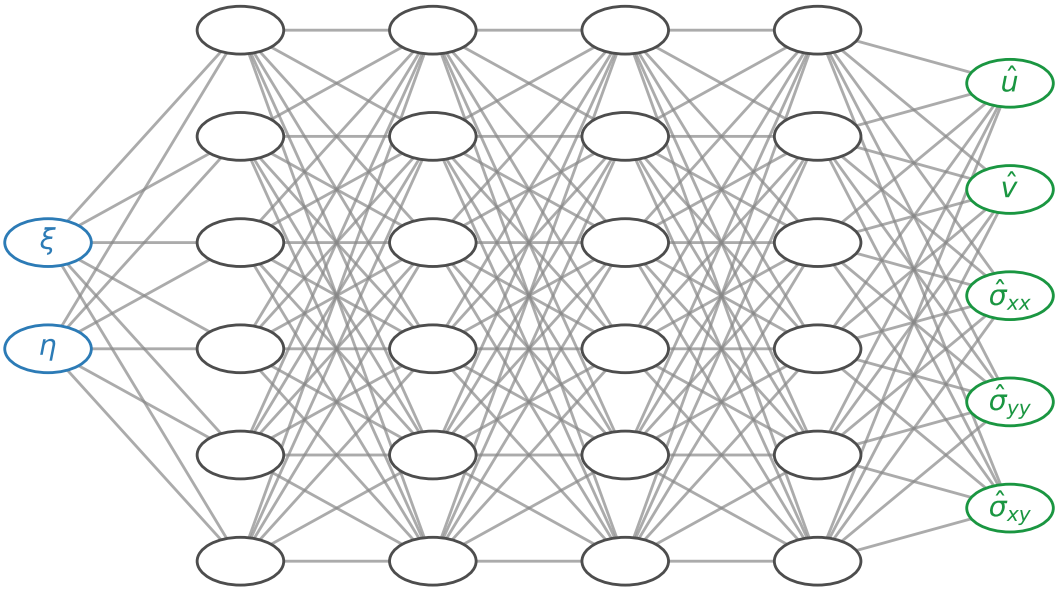
$$R_{\mathrm{comp}} = \nabla^2(\hat{\sigma}_{xx} + \hat{\sigma}_{yy}), \tag{7.82}$$

natomiast trzy residua konstytutywne wymuszają zgodność naprężeń sieciowych ze związkiem konstytutywnym:

$$\boldsymbol{R}_\sigma = \frac{\mathbf{C}\cdot\boldsymbol{\varepsilon}(\hat{\mathbf{u}})}{\sigma_0} - \hat{\boldsymbol{\sigma}}, \tag{7.83}$$

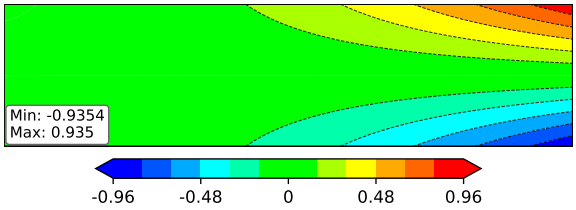
gdzie $\hat{\boldsymbol{\sigma}} \to (\hat{\sigma}_{xx}, \hat{\sigma}_{yy}, \hat{\sigma}_{xy})$ oznacza bezpośrednie wyjścia naprężeniowe sieci.

Zastosowano architekturę FNN z czterema warstwami ukrytymi po 64 neurony i pięcioma wyjściami. Schemat architektury przedstawiono na rys. 7.20.

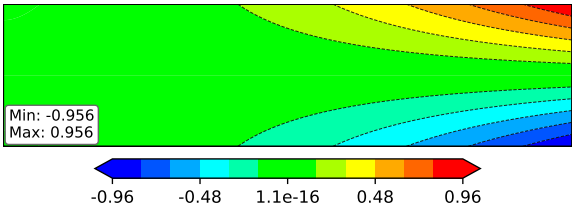


Rysunek 7.20. Schemat architektury sieci FNN w wariancie W2

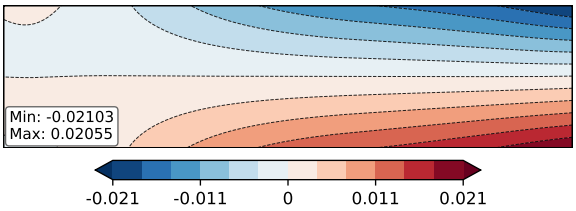
Porównanie przewidywanych pól przemieszczeń i naprężeń z rozwiązaniem referencyjnym (TS) przedstawiono na rys. 7.21–7.25.



(a) $\sigma_{xx}^{\mathrm{PINN^{W2}}}$ [kN/cm^2]

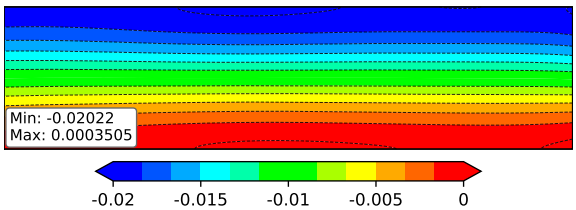


(b) $\sigma_{xx}^{\mathrm{TS}}$ [kN/cm^2]

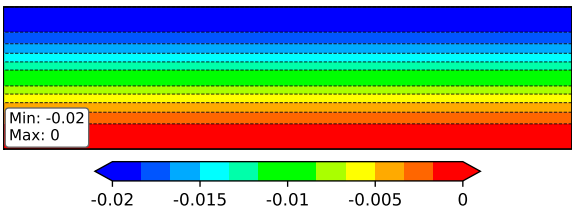


(c) Błąd $\Delta\sigma_{xx}$ [kN/cm^2]

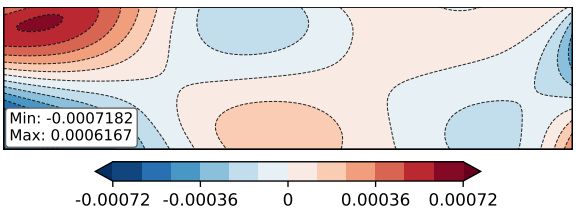
Rysunek 7.21. Wykresy warstwicowe składowej naprężenia σ_{xx} [kN/cm^2]: a) rozwiązanie z PINN w wariancie W2, b) rozwiązanie z teorii sprężystości (TS), c) błąd bezwzględny $\Delta\sigma_{xx} = \sigma_{xx}^{\mathrm{PINN^{W2}}} - \sigma_{xx}^{\mathrm{TS}}$



(a) $\sigma_{yy}^{\mathrm{PINN^{W2}}}$ [kN/cm^2]

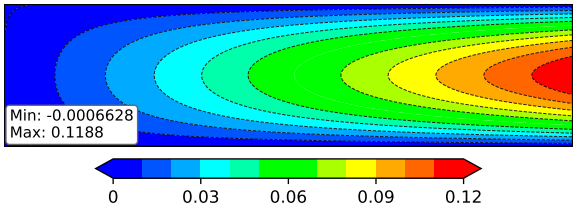


(b) $\sigma_{yy}^{\mathrm{TS}}$ [kN/cm^2]

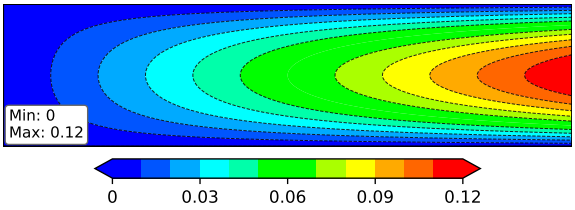


(c) Błąd $\Delta\sigma_{yy}$ [kN/cm^2]

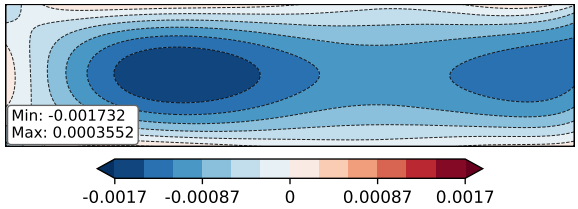
Rysunek 7.22. Wykresy warstwicowe składowej naprężenia σ_{yy} [kN/cm^2]: a) rozwiązanie z PINN w wariancie W2, b) rozwiązanie z teorii sprężystości (TS), c) błąd bezwzględny $\Delta\sigma_{yy} = \sigma_{yy}^{\mathrm{PINN^{W2}}} - \sigma_{yy}^{\mathrm{TS}}$



(a) $\sigma_{xy}^{\mathrm{PINN}^{\mathrm{W2}}}$ [kN/cm^2]

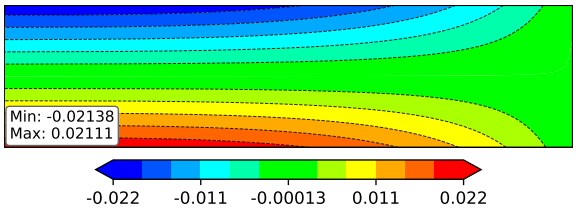


(b) $\sigma_{xy}^{\mathrm{TS}}$ [kN/cm^2]

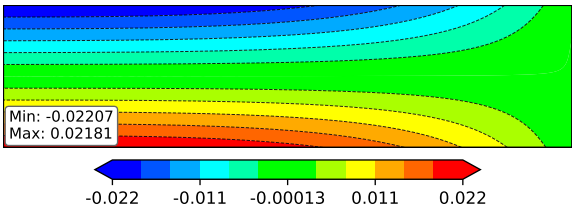


(c) Błąd $\Delta\sigma_{xy}$ [kN/cm^2]

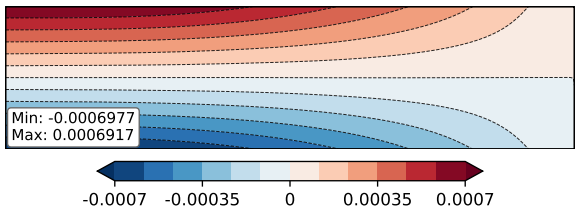
Rysunek 7.23. Wykresy warstwicowe składowej naprężenia σ_{xy} [kN/cm^2]: a) rozwiązanie z PINN w wariancie W2, b) rozwiązanie z teorii sprężystości (TS), c) błąd bezwzględny $\Delta\sigma_{xy} = \sigma_{xy}^{\mathrm{PINN}^{\mathrm{W2}}} - \sigma_{xy}^{\mathrm{TS}}$



(a) $u_x^{\mathrm{PINN}^{\mathrm{W2}}}$ [cm]

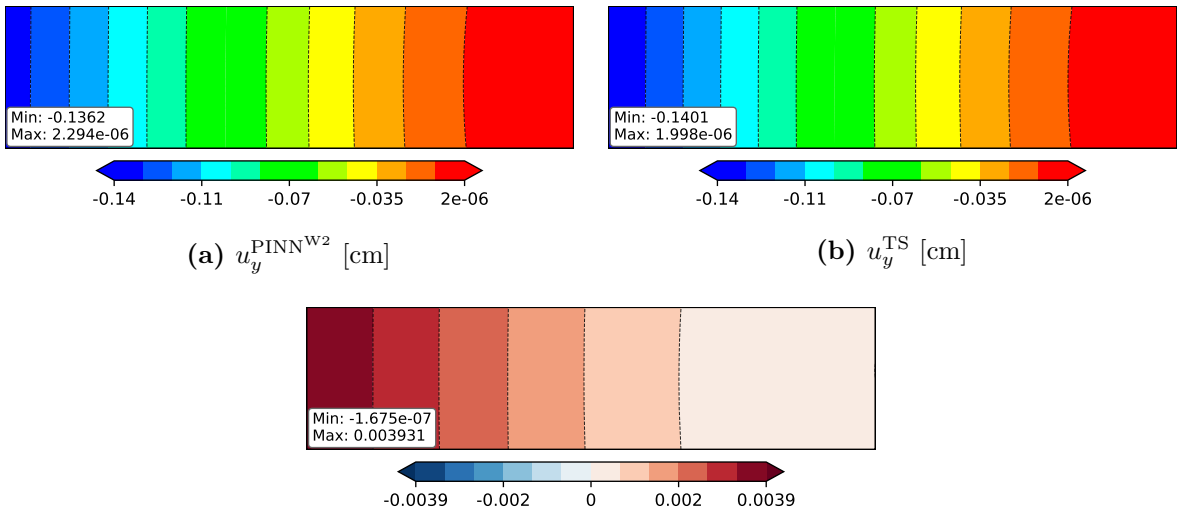


(b) u_x^{TS} [cm]



(c) Błąd Δu_x [cm]

Rysunek 7.24. Wykresy warstwicowe przemieszczenia poziomego u_x [cm]: a) rozwiązanie z PINN w wariancie W2, b) rozwiązanie z teorii sprężystości (TS), c) błąd bezwzględny $\Delta u_x = u_x^{\mathrm{PINN}^{\mathrm{W2}}} - u_x^{\mathrm{TS}}$



(a) $u_y^{\mathrm{PINN}^{\mathrm{W2}}}$ [cm] **(b)** u_y^{TS} [cm]

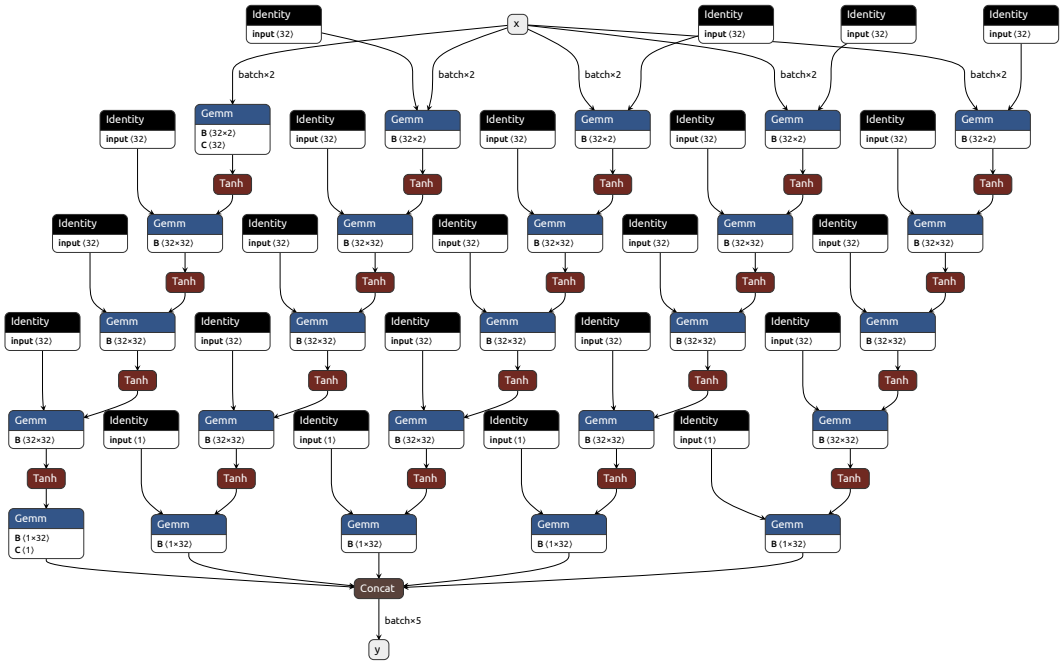
(c) Błąd Δu_y [cm]

Rysunek 7.25. Wykresy warstwicowe przemieszczenia pionowego u_y [cm]: a) rozwiązanie z PINN w wariancie W2, b) rozwiązanie z teorii sprężystości (TS), c) błąd bezwzględny $\Delta u_y = u_y^{\mathrm{PINN}^{\mathrm{W2}}} - u_y^{\mathrm{TS}}$

Wariant W3 — PFNN, sformułowanie mieszane, miękkie warunki brzegowe

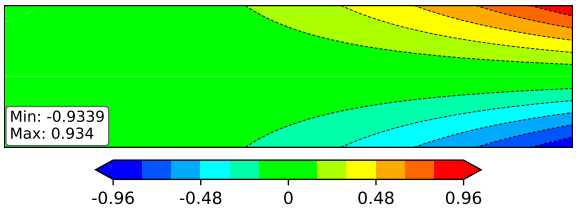
Wariant W3 wykorzystuje to samo sformułowanie mieszane i miękkie warunki brzegowe co wariant W2. Jedyną różnicą jest zastosowanie architektury PFNN (*Parallel Fully-Connected Network*), w której każda z pięciu składowych wyjściowych ($\hat{u}$, $\hat{v}$, $\hat{\sigma}_{xx}$, $\hat{\sigma}_{yy}$, $\hat{\sigma}_{xy}$) aproksymowana jest przez odrębną podsieć (gałąź) o strukturze [2, 32, 32, 32, 32, 1]. Gałęzie nie są ze sobą sprzężone — nie stosowano połączeń krzyżowych ani ponownego scalania reprezentacji pomiędzy warstwami. Architekturę PFNN zastosowano wzorując się na przykładzie z dokumentacji biblioteki DeepXDE [41], w którym wykorzystano ją do rozwiązania zadania teorii sprężystości. Należy jednak zauważyć, że w rozpatrywanym problemie pola przemieszczeń i naprężeń są ze sobą ściśle powiązane poprzez związek konstytutywny, co podważa zasadność stosowania niezależnych gałęzi. Weryfikację tej hipotezy — tj. czy architektura PFNN przynosi rzeczywistą korzyść w porównaniu z FNN dla tego typu zadań — przedstawiono w dalszej części pracy na podstawie porównania wyników wariantów W2 i W3. Schemat zastosowanej architektury PFNN przedstawiono na rys. 7.26. Grafika została wygenerowana automatycznie na podstawie wytrenowanego modelu przy użyciu narzędzia Netron [137] i odzwierciedla rzeczywistą strukturę obliczeniową sieci. Bloki typu `Gemm` odpowiadają warstwom sieci wielowarstwowej (operacja mnożenia macierzy wag przez wektor wejściowy wraz z dodaniem wektora wyrazów wolnych), natomiast

węzły `Identity` reprezentują operacje identyczności, wprowadzane przez framework w celu organizacji grafu obliczeniowego i nie wpływają na wynik predykcji.

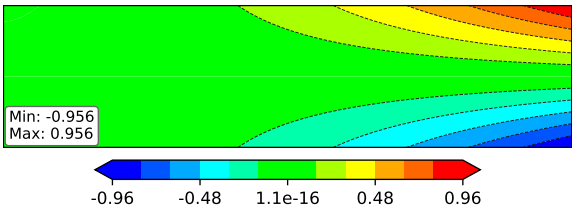


Rysunek 7.26. Schemat architektury zastosowanej sieci PFNN w wariancie W3

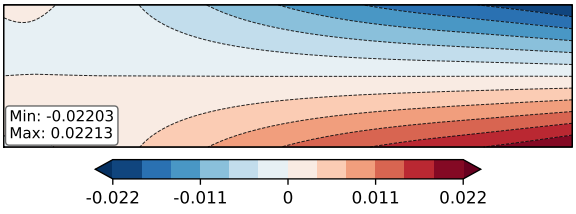
Porównanie przewidywanych pól przemieszczeń i naprężeń z rozwiązaniem referencyjnym (TS) przedstawiono na rys. 7.27–7.31.



(a) $\sigma_{xx}^{\mathrm{PINN^{W3}}}$ [kN/cm^2]

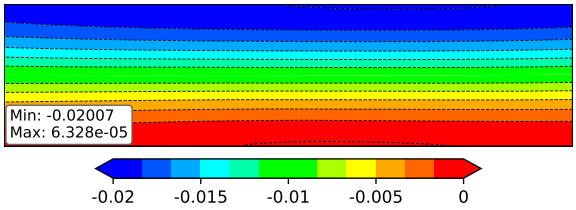


(b) $\sigma_{xx}^{\mathrm{TS}}$ [kN/cm^2]

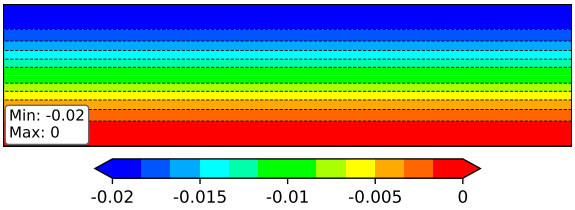


(c) Błąd $\Delta\sigma_{xx}$ [kN/cm^2]

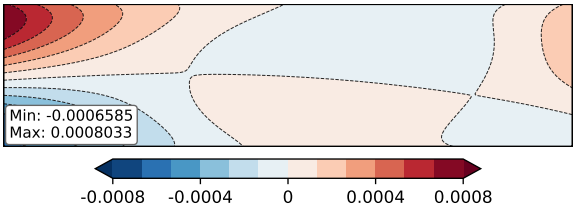
Rysunek 7.27. Wykresy warstwicowe składowej naprężenia σ_{xx} [kN/cm^2]: a) rozwiązanie z PINN w wariancie W3, b) rozwiązanie z teorii sprężystości (TS), c) błąd bezwzględny $\Delta\sigma_{xx} = \sigma_{xx}^{\mathrm{PINN^{W3}}} - \sigma_{xx}^{\mathrm{TS}}$



(a) $\sigma_{yy}^{\mathrm{PINN}^{\mathrm{W3}}}$ [kN/cm^2]

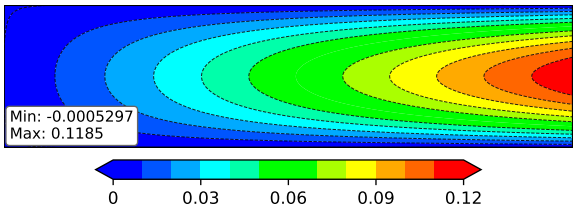


(b) $\sigma_{yy}^{\mathrm{TS}}$ [kN/cm^2]

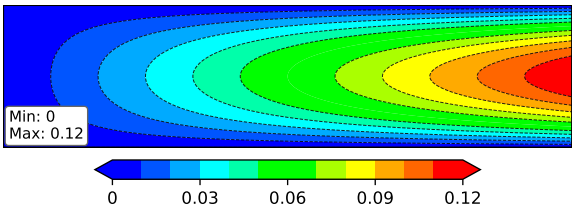


(c) Błąd $\Delta\sigma_{yy}$ [kN/cm^2]

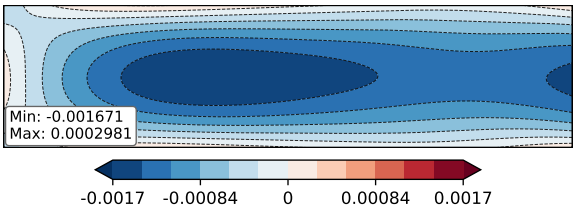
Rysunek 7.28. Wykresy warstwicowe składowej naprężenia σ_{yy} [kN/cm^2]: a) rozwiązanie z PINN w wariancie W3, b) rozwiązanie z teorii sprężystości (TS), c) błąd bezwzględny $\Delta\sigma_{yy} = \sigma_{yy}^{\mathrm{PINN}^{\mathrm{W3}}} - \sigma_{yy}^{\mathrm{TS}}$



(a) $\sigma_{xy}^{\mathrm{PINN}^{\mathrm{W3}}}$ [kN/cm^2]

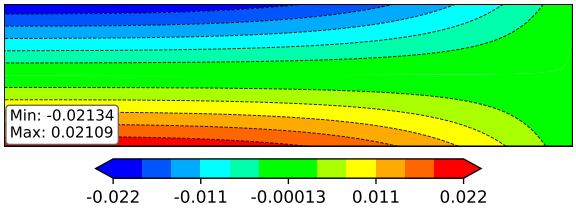


(b) $\sigma_{xy}^{\mathrm{TS}}$ [kN/cm^2]

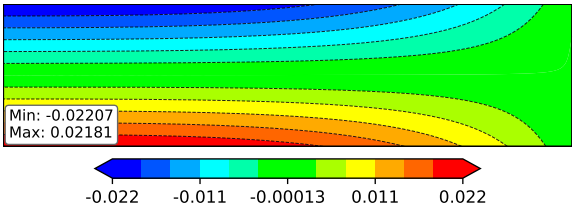


(c) Błąd $\Delta\sigma_{xy}$ [kN/cm^2]

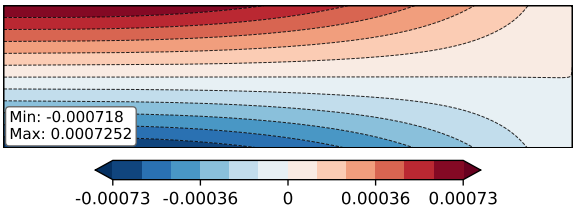
Rysunek 7.29. Wykresy warstwicowe składowej naprężenia σ_{xy} [kN/cm^2]: a) rozwiązanie z PINN w wariancie W3, b) rozwiązanie z teorii sprężystości (TS), c) błąd bezwzględny $\Delta\sigma_{xy} = \sigma_{xy}^{\mathrm{PINN}^{\mathrm{W3}}} - \sigma_{xy}^{\mathrm{TS}}$



(a) $u_x^{\mathrm{PINN}^{\mathrm{W3}}}$ [cm]

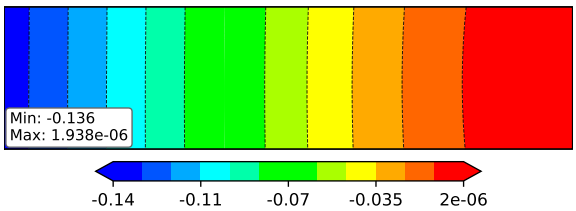


(b) u_x^{TS} [cm]

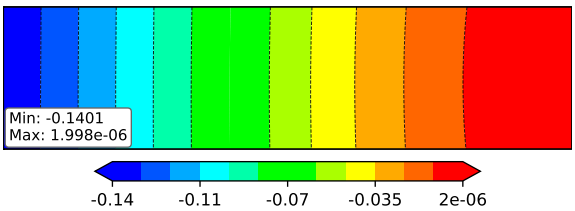


(c) Błąd Δu_x [cm]

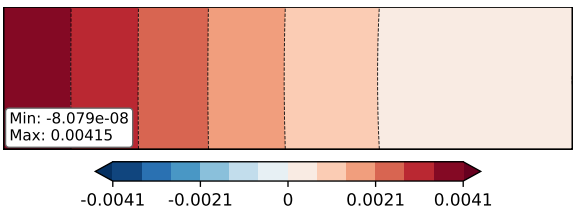
Rysunek 7.30. Wykresy warstwicowe przemieszczenia poziomego u_x [cm]: a) rozwiązanie z PINN w wariancie W3, b) rozwiązanie z teorii sprężystości (TS), c) błąd bezwzględny $\Delta u_x = u_x^{\mathrm{PINN}^{\mathrm{W3}}} - u_x^{\mathrm{TS}}$



(a) $u_y^{\mathrm{PINN}^{\mathrm{W3}}}$ [cm]



(b) u_y^{TS} [cm]



(c) Błąd Δu_y [cm]

Rysunek 7.31. Wykresy warstwicowe przemieszczenia pionowego u_y [cm]: a) rozwiązanie z PINN w wariancie W3, b) rozwiązanie z teorii sprężystości (TS), c) błąd bezwzględny $\Delta u_y = u_y^{\mathrm{PINN}^{\mathrm{W3}}} - u_y^{\mathrm{TS}}$

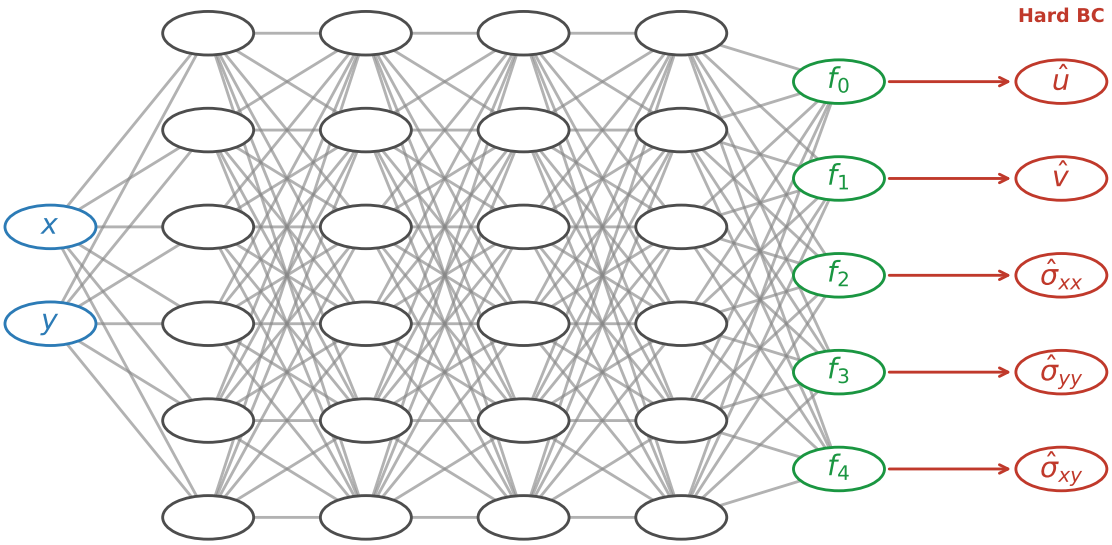
Wariant W4 — FNN, sformułowanie, twarde warunki brzegowe

Wariant W4 wykorzystuje to samo sformułowanie mieszane i architekturę FNN co wariant W2. Różnica polega na sposobie wymuszania warunków brzegowych — zastosowano twarde warunki brzegowe (*hard BCs*), polegające na transformacji wyjść sieci, tak aby

warunki brzegowe były spełnione z definicji, niezależnie od wartości parametrów sieci. Transformację zdefiniowano jako:

$$\begin{aligned} \hat{u} &\to f_0\,(1-\xi), \\ \hat{v} &\to f_1\,(1-\xi), \\ \hat{\sigma}_{xx} &\to f_2\,\xi, \\ \hat{\sigma}_{yy} &\to -\hat{q}\left(\eta+\tfrac{1}{2}\right)+f_3\left(\eta+\tfrac{1}{2}\right)\left(\eta-\tfrac{1}{2}\right), \\ \hat{\sigma}_{xy} &\to f_4\,\xi\left(\eta+\tfrac{1}{2}\right)\left(\eta-\tfrac{1}{2}\right), \end{aligned} \tag{7.84}$$

gdzie f_i oznaczają surowe wyjścia sieci, a $\hat{q} = q/\sigma_0$. Mnożniki zerują się na odpowiednich brzegach, wymuszając: $\hat{u} = 0$ i $\hat{v} = 0$ przy $\xi = 1$ (utwierdzenie na prawym brzegu), $\hat{\sigma}_{xx} = 0$ i $\hat{\sigma}_{xy} = 0$ przy $\xi = 0$ (wolny lewy brzeg), $\hat{\sigma}_{yy} = 0$ i $\hat{\sigma}_{xy} = 0$ przy $\eta = -\frac{1}{2}$ (wolna dolna krawędź) i $\hat{\sigma}_{yy} = -\hat{q}$ i $\hat{\sigma}_{xy} = 0$ przy $\eta = \frac{1}{2}$ (obciążenie na górnej krawędzi). W konsekwencji składowa L_{BC} w funkcji kosztu jest pusta — sieć spełnia warunki brzegowe z definicji. Schemat architektury przedstawiono na rys. 7.32.



Rysunek 7.32. Schemat architektury sieci FNN w wariancie W4

Uzyskanie zbieżności wariantu W4 wymagało stopniowego dostrajania hiperparametrów treningowych. W konfiguracji początkowej przy kroku uczenia 0.001 i 1000 punktów

względny błąd ϵ_{ℓ_2} wynosił 0.61, co świadczy o braku zbieżności, nie zaś jedynie o niedostatecznej dokładności.

W pierwszej kolejności zwiększono liczbę punktów residuum z 1000 do 8000. Analogicznie jak w MES, gdzie zagęszczenie siatki jest konieczne do uchwycenia koncentracji naprężeń w pobliżu utwierdzenia, tak w PINN wystarczająca gęstość punktów residuum warunkuje zdolność sieci do odwzorowania lokalnych gradientów pola rozwiązania. Błąd ϵ_{ℓ_2} spadł do 0.28.

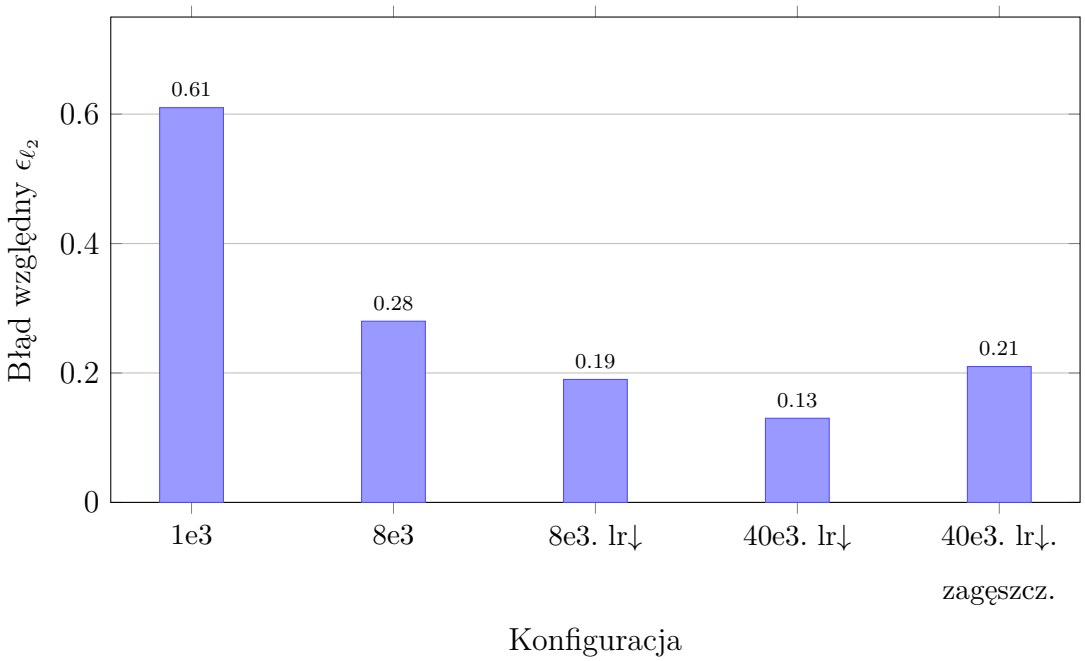
Następnie zmniejszono krok uczenia z 0.001 do 0.0005. W wariantach z miękkimi warunkami brzegowymi funkcja kosztu zawiera jawny składnik L_{BC}, który dostarcza optymalizatorowi wyraźny sygnał we wczesnych fazach treningu — sieć szybko uczy się spełniać warunki brzegowe, co nadaje wyjściom sensowny kształt i stabilizuje dalszą optymalizację. W wariancie W4 tego składnika nie ma, a całość informacji o rozwiązaniu pochodzi wyłącznie z residuum PDE, które jest trudniejszym sygnałem treningowym. Ponadto surowe wyjścia sieci f_i są mnożone przez funkcje współrzędnych (np. $\hat{\sigma}_{xy} = f_4\,\xi\,(\eta + \frac{1}{2})(\eta - \frac{1}{2})$), co przy obliczaniu gradientów funkcji kosztu po parametrach sieci — na mocy reguły łańcuchowej — powoduje, że gradienty stają się bardziej zmienne, zależąc nie tylko od f_i, ale też od wartości współrzędnych w danym punkcie. Przy zbyt dużym kroku uczenia te bardziej zmienne gradienty prowadzą do nadmiernych skoków parametrów między kolejnymi iteracjami. Zmniejszenie kroku uczenia ogranicza te skoki i stabilizuje przebieg treningu. Błąd ϵ_{ℓ_2} spadł do 0.19.

Kolejnym krokiem było zwiększenie liczby punktów residuum do 40 000. Rozwiązanie referencyjne MES wykorzystuje ok. 40 000 elementów skończonych ze zredukowanym całkowaniem, co daje porównywalną liczbę punktów ewaluacji równań — przyjęto zatem analogiczną gęstość próbkowania domeny. Błąd ϵ_{ℓ_2} spadł do 0.14.

Na końcu dodano dodatkowe punkty residuum: do 40 000 punktów równomiernie rozmieszczonych w domenie dodano po 5000 punktów w dwóch strefach narożnych przy utwierdzeniu — $\xi \in [0.70;\, 1.0]$, $\eta \in [0.20;\, 0.5]$ oraz $\xi \in [0.70;\, 1.0]$, $\eta \in [-0.5;\, -0.20]$. Zagęszczenie w tych obszarach wynika z tego, że w pobliżu utwierdzenia występują najwyższe gradienty naprężeń i przemieszczeń — jest to strefa, w której sieć najtrudniej odwzorowuje rozwiązanie i w której błąd residuum PDE jest z reguły największy.

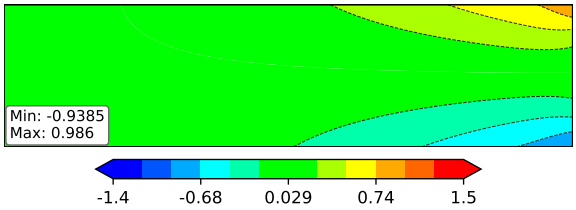
Przebieg względnego błędu ϵ_{ℓ_2} w kolejnych konfiguracjach przedstawiono na rys. 7.33. Błąd obliczano między rozwiązaniem PINN a referencyjnym rozwiązaniem MES na zbiorze punktów testowych.

Dodanie zagęszczonych punktów w strefach narożnych nie przyniosło dalszej poprawy błędu ϵ_{ℓ_2}. Najbardziej prawdopodobną przyczyną jest to, że wprowadzenie 10 000 dodatkowych punktów skoncentrowanych w wąskim obszarze przy 40 000 punktów w całej domenie sprawia, iż około 20% sygnału treningowego pochodzi ze stosunkowo niewielkiej strefy. Wang i in. [136] wykazali, że poszczególne składniki funkcji kosztu PINN mogą zbiegać z bardzo różnymi prędkościami, prowadząc do braku zbieżności treningu. Nadreprezentacja punktów z okolicy utwierdzenia działa analogicznie — optymalizator może poprawiać dopasowanie w strefie narożnej kosztem pogorszenia w pozostałych częściach domeny, w efekcie czego globalny błąd ϵ_{ℓ_2} nie maleje. Dodatkowym czynnikiem jest to, że rozkład zagęszczonych punktów został ustalony z góry, a nie na podstawie bieżących wartości residuum. Wu i in. [138] pokazali, że dynamiczne metody próbkowania (RAD, RAR-D) istotnie poprawiają dokładność PINN, ponieważ kierują punkty tam, gdzie sieć faktycznie popełnia największe błędy.

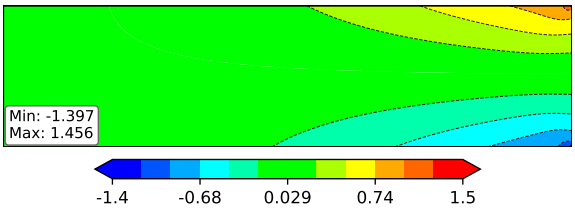


Rysunek 7.33. Względny błąd ϵ_{ℓ_2} wariantu W4 w funkcji kolejnych modyfikacji hiperparametrów: zwiększenie liczby punktów residuum (1 000 → 8 000 → 40 000), zmniejszenie kroku uczenia (lr↓) oraz zagęszczenie punktów w strefach narożnych przy utwierdzeniu

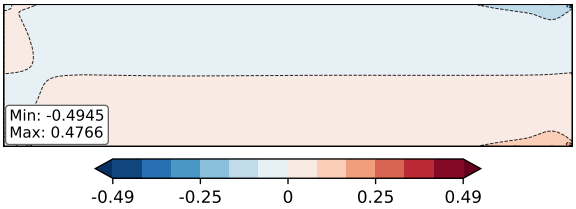
Porównanie przewidywanych pól przemieszczeń i naprężeń z rozwiązaniem referencyjnym (MES) przedstawiono na rys. 7.34–7.38.



(a) $\sigma_{xx}^{\mathrm{PINN^{W4}}}$ [kN/cm^2]

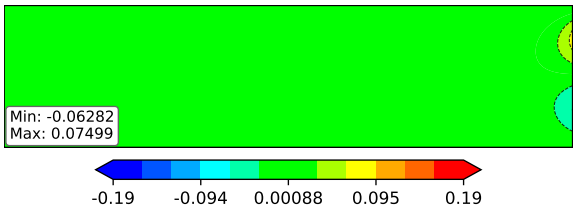


(b) $\sigma_{xx}^{\mathrm{MES}}$ [kN/cm^2]

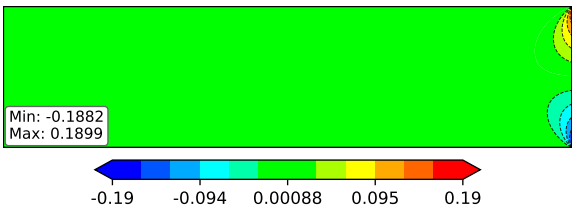


(c) Błąd $\Delta\sigma_{xx}$ [kN/cm^2]

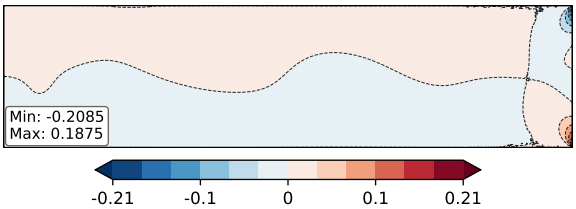
Rysunek 7.34. Wykresy warstwicowe składowej naprężenia σ_{xx} [kN/cm^2]: a) rozwiązanie z PINN w wariancie W4, b) rozwiązanie z MES, c) błąd bezwzględny $\Delta\sigma_{xx} = \sigma_{xx}^{\mathrm{PINN^{W4}}} - \sigma_{xx}^{\mathrm{MES}}$



(a) $\sigma_{yy}^{\mathrm{PINN^{W4}}}$ [kN/cm^2]

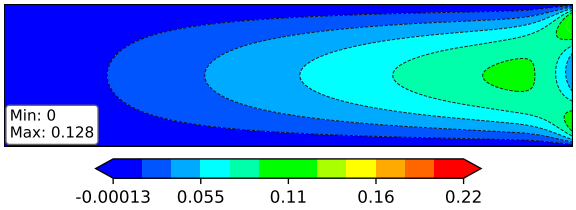


(b) $\sigma_{yy}^{\mathrm{MES}}$ [kN/cm^2]

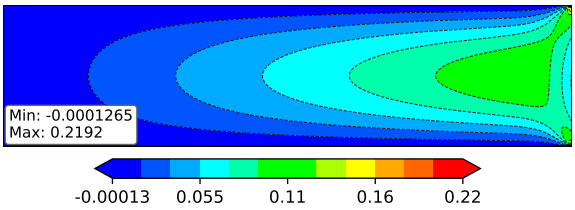


(c) Błąd $\Delta\sigma_{yy}$ [kN/cm^2]

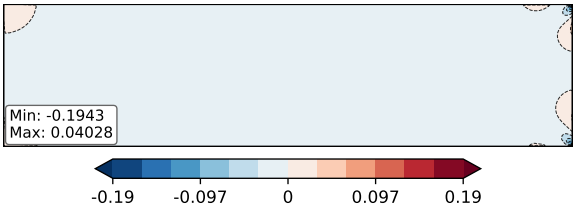
Rysunek 7.35. Wykresy warstwicowe składowej naprężenia σ_{yy} [kN/cm^2]: a) rozwiązanie z PINN w wariancie W4, b) rozwiązanie z MES, c) błąd bezwzględny $\Delta\sigma_{yy} = \sigma_{yy}^{\mathrm{PINN^{W4}}} - \sigma_{yy}^{\mathrm{MES}}$



(a) $\sigma_{xy}^{\mathrm{PINN^{W4}}}$ [kN/cm^2]

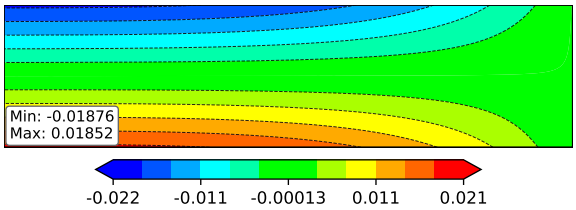


(b) $\sigma_{xy}^{\mathrm{MES}}$ [kN/cm^2]

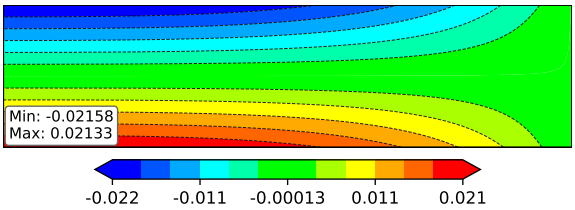


(c) Błąd $\Delta\sigma_{xy}$ [kN/cm^2]

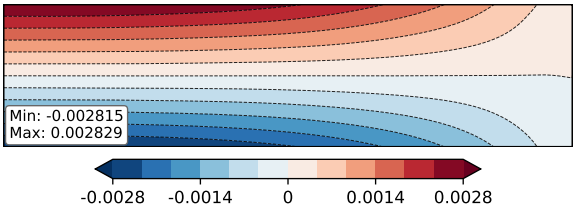
Rysunek 7.36. Wykresy warstwicowe składowej naprężenia σ_{xy} [kN/cm^2]: a) rozwiązanie z PINN w wariancie W4, b) rozwiązanie z MES, c) błąd bezwzględny $\Delta\sigma_{xy} = \sigma_{xy}^{\mathrm{PINN^{W4}}} - \sigma_{xy}^{\mathrm{MES}}$



(a) $u_x^{\mathrm{PINN^{W4}}}$ [cm]

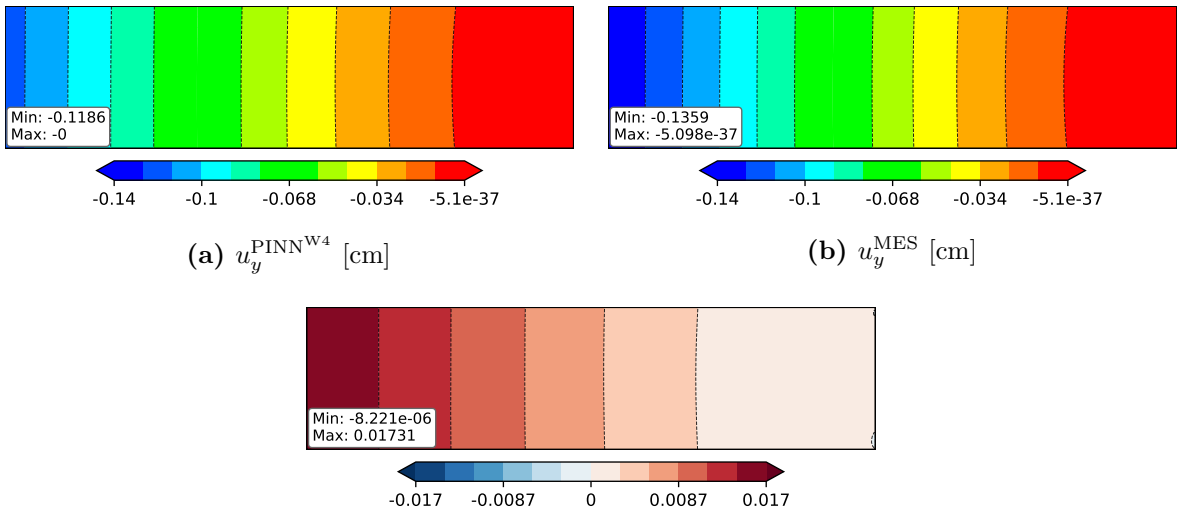


(b) u_x^{MES} [cm]



(c) Błąd Δu_x [cm]

Rysunek 7.37. Wykresy warstwicowe przemieszczenia poziomego u_x [cm]: a) rozwiązanie z PINN w wariancie W4, b) rozwiązanie z MES, c) błąd bezwzględny $\Delta u_x = u_x^{\mathrm{PINN^{W4}}} - u_x^{\mathrm{MES}}$



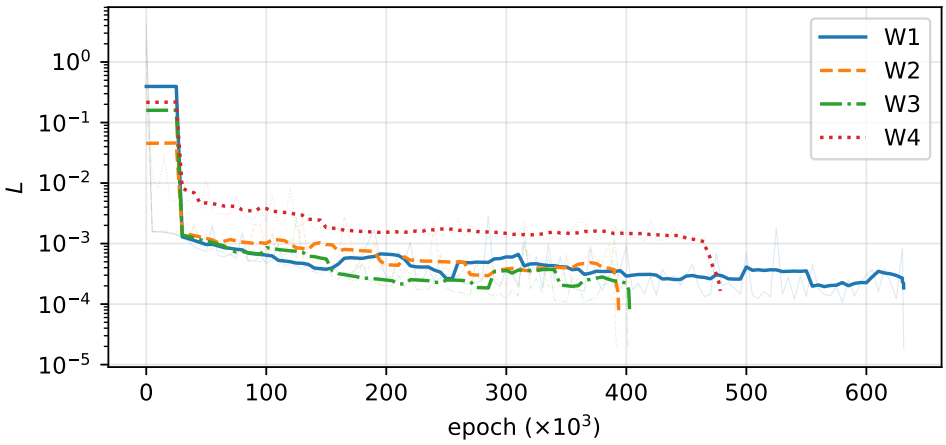
(a) $u_y^{\mathrm{PINN}^{\mathrm{W4}}}$ [cm] **(b)** u_y^{MES} [cm]

(c) Błąd Δu_y [cm]

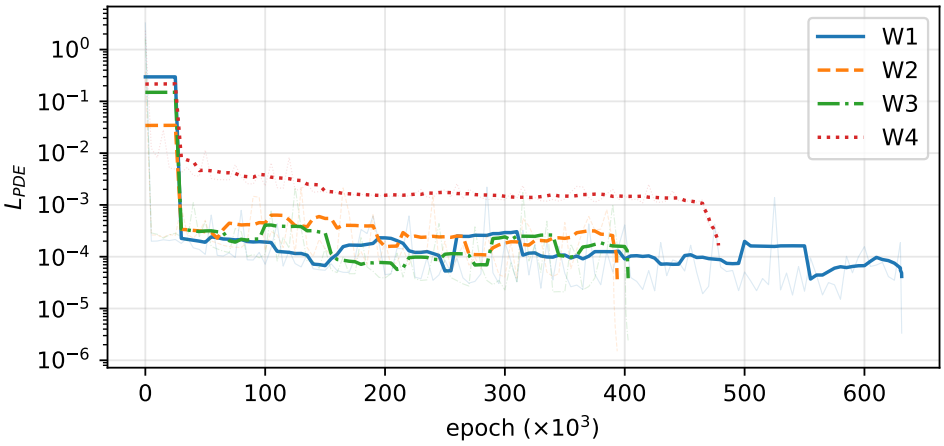
Rysunek 7.38. Wykresy warstwicowe przemieszczenia pionowego u_y [cm]: a) rozwiązanie z PINN w wariancie W4, b) rozwiązanie z MES, c) błąd bezwzględny $\Delta u_y = u_y^{\mathrm{PINN}^{\mathrm{W4}}} - u_y^{\mathrm{MES}}$

Wyniki

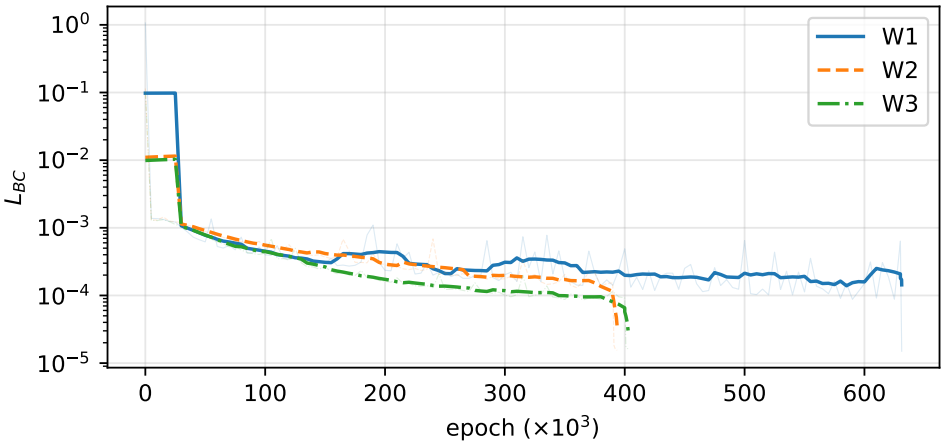
Porównanie przewidywanych pól przemieszczeń i naprężeń ze wszystkich wariantów z rozwiązaniem referencyjnym (TS, a w wariancie W4 – MES) przedstawiono na rys. 7.15–7.38. Przebiegi zbieżności funkcji kosztu dla wariantów W1–W4 zestawiono na rys. 7.39–7.41. Zbiorcze zestawienie wyników przedstawiono w tabeli 7.3. Wszystkie obliczenia przeprowadzono na karcie graficznej NVIDIA RTX A6000 (48 GB VRAM) z wykorzystaniem biblioteki DeepXDE w wersji 1.15.0 [131] z backendem PyTorch [42].



Rysunek 7.39. Porównanie sumarycznej funkcji kosztu L dla wariantów W1–W4



Rysunek 7.40. Porównanie składowej L_{PDE} dla wariantów W1–W4



Rysunek 7.41. Porównanie składowej L_{BC} dla wariantów W1–W3 (wariant W4 nie posiada składowej L_{BC})

Tabela 7.3. Zbiorcze zestawienie wyników wariantów W1–W4: względny błąd ϵ_{ℓ_2} przemieszczeń i naprężeń względem rozwiązania referencyjnego (TS/MES), liczba iteracji fazy Adam oraz czas obliczeń

	W1	**W2**	**W3**	**W4**
$\epsilon_{\ell_2}(\mathbf{u})$ [%]	3.82	2.75	2.91	12.56
$\epsilon_{\ell_2}(\boldsymbol{\sigma})$ [%]	—	2.84	3.00	12.60
ϵ_{ℓ_2}(całk.) [%]	3.82	2.76	2.92	12.56
Iteracje Adam [$\times 10^3$]	630	390	400	465
Iteracje L-BFGS	1092	3723	2850	13206
Czas Adam [s]	11793	3678	9999	8732
Czas L-BFGS [s]	26	188	184	328

Warianty W1–W3 osiągnęły zbliżony poziom błędu względnego ϵ_{ℓ_2} rzędu 2.8–3.8 % dla przemieszczeń i 2.8–3.0 % dla naprężeń względem rozwiązania TS. Najniższy błąd uzyskał wariant W2 ($\epsilon_{\ell_2}(\mathbf{u}) = 2.75\,\%$, $\epsilon_{\ell_2}(\boldsymbol{\sigma}) = 2.84\,\%$), w którym pola naprężeń są aproksymowane jawnie przez dodatkowe wyjścia sieci.

Wariant W1 (formulacja w przemieszczeniach) wymagał najdłuższego treningu fazą Adam (630×10^3 iteracji, 11 793 s), jednak uzyskał błąd jedynie nieznacznie wyższy ($\epsilon_{\ell_2}(\mathbf{u}) = 3.82\,\%$). Dłuższy czas zbieżności wynika z faktu, że sieć przewiduje tylko dwa pola przemieszczeń, a naprężenia są wyprowadzane pośrednio przez różniczkowanie i związek konstytutywny — podejście to jest bardziej spójne z teorią sprężystości, lecz wymaga od optymalizatora jednoczesnego dopasowania wartości przemieszczeń i ich pochodnych wyższego rzędu.

Wariant W3 (PFNN) osiągnął błąd porównywalny z W2 ($\epsilon_{\ell_2}(\mathbf{u}) = 2.91\,\%$, $\epsilon_{\ell_2}(\boldsymbol{\sigma}) = 3.00\,\%$), jednak przy znacznie dłuższym czasie obliczeń (10 182 s wobec 3 866 s dla W2). W architekturze PFNN każde pole wyjściowe jest aproksymowane przez odrębną podsieć, co eliminuje współdzielenie cech w warstwach ukrytych między komponentami rozwiązania. Taki podział nie ma uzasadnienia fizycznego — składowe przemieszczeń i naprężeń są ze sobą ściśle powiązane przez równania równowagi i związek konstytutywny, a ich niezależna aproksymacja uniemożliwia sieci wykorzystanie tych zależności. W efekcie

architektura PFNN nie przynosi korzyści dokładnościowych względem standardowej sieci FNN (W2), jednocześnie zwiększając koszt obliczeniowy.

Wariant W4 (twarde warunki brzegowe) osiągnął błąd $\epsilon_{\ell_2}(\text{całk.}) = 12.56\,\%$, co jest wynikiem istotnie wyższym niż w wariantach W1–W3. Różnica ta wynika jednak przede wszystkim ze zmiany rozwiązania referencyjnego — wariant W4 jako jedyny jest porównywany z MES, a nie z rozwiązaniem TS. Zarówno w PINN (W4), jak i w MES warunki brzegowe na utwierdzeniu narzucono w sposób silny, jednak w teorii sprężystości, na której oparto rozwiązanie referencyjne, zastosowano słabe warunki brzegowe, co eliminuje lokalne koncentracje naprężeń obecne w rozwiązaniu MES. W efekcie charakter warstwic w wariancie W4 jest jakościowo zgodny z MES, natomiast wartości ekstremalne są bliższe rozwiązaniu TS. Pomimo eliminacji składowej L_{BC} z funkcji kosztu (rys. 7.41), wariant ten wymagał największej liczby iteracji L-BFGS (13 206).

Pola naprężeń σ_{xx}, σ_{yy}, σ_{xy} oraz przemieszczenia u_x, u_y są jakościowo poprawnie odwzorowane we wszystkich wariantach. Największe błędy bezwzględne obserwuje się w okolicach utwierdzenia, gdzie gradienty pól są najwyższe, co jest charakterystyczne zarówno dla Metody Elementów Skończonych z rzadką dyskretyzacją, jak i dla sieci PINN z niewystarczającym zagęszczeniem punktów residuum.

Dodatkowo dla wariantu W2 przeprowadzono dwa eksperymenty pomocnicze mające na celu zbadanie wpływu fazy Adam na końcową dokładność modelu. Obserwacja przebiegu funkcji kosztu (rys. 7.39) wskazuje, że najintensywniejszy spadek funkcji kosztu następuje w pierwszych $\sim 30 \times 10^3$ iteracjach Adama, po czym dalszy postęp jest niewielki aż do momentu włączenia optymalizatora L-BFGS, który ponownie znacząco redukuje wartość funkcji kosztu. W celu weryfikacji tej obserwacji w pierwszym eksperymencie całkowicie pominięto fazę Adam i uruchomiono trening wyłącznie z optymalizatorem L-BFGS — sieć osiągnęła błąd $\epsilon_{\ell_2}(\text{całk.}) = 3.72\,\%$ po zaledwie 6 834 iteracjach w czasie $\sim 94\,\text{s}$, co stanowi wynik porównywalny z pełnym treningiem W2 (2.76 %), uzyskany jednak w czasie krótszym o ponad dwa rzędy wielkości. W drugim eksperymencie fazę Adam ograniczono do 30×10^3 iteracji, po czym kontynuowano trening optymalizatorem L-BFGS — uzyskano błąd $\epsilon_{\ell_2}(\text{całk.}) = 5.01\,\%$ po 5 541 iteracjach L-BFGS w łącznym czasie $\sim 325\,\text{s}$. Paradoksalnie skrócenie fazy Adam pogorszyło końcowy wynik względem wariantu bez Adama, co może wskazywać, że częściowy trening Adamem prowadzi sieć do obszaru przestrzeni parametrów, z którego L-BFGS trudniej osiąga globalne minimum.

Niezależnie od tego oba eksperymenty potwierdzają, że dla rozpatrywanego zadania sam optymalizator L-BFGS jest w stanie uzyskać satysfakcjonującą dokładność w czasie rzędu minut — tak krótki czas obliczeń sugeruje, że analogiczne eksperymenty byłyby wykonalne również na konsumenckich kartach graficznych.

7.4.4. Wnioski

Na podstawie przeprowadzonych obliczeń dla trzech zadań — porównania implementacji w czystym PyTorch z biblioteką DeepXDE, zagadnienia belkowego (zadanie 1D) oraz tarczy w płaskim stanie naprężenia (zadanie 2D) — sformułowano następujące wnioski.

Biblioteka DeepXDE stanowi wygodną nakładkę na framework PyTorch, usprawniającą definiowanie geometrii, warunków brzegowych, residuów PDE oraz strategii próbkowania punktów kolokacyjnych. Porównanie z własną implementacją w czystym PyTorch wykazało, że obie ścieżki prowadzą do tożsamych wyników, jednak DeepXDE istotnie redukuje ilość kodu i czas potrzebny na przygotowanie eksperymentu, co czyni ją praktycznym narzędziem do prototypowania modeli PINN.

Kluczowym wnioskiem płynącym z zadań belkowego i tarczowego jest zróżnicowana wrażliwość sieci na skalowanie danych wejściowych oraz wyjściowych w zależności od wymiaru zadania. W zagadnieniu 1D (belka) wystarczające okazało się znormalizowanie współrzędnej przestrzennej do przedziału [0, 1] — skalowanie ugięcia nie było konieczne, ponieważ sieć samodzielnie radziła sobie z aproksymacją pola o jednorodnym rzędzie wielkości. W zadaniu 2D normalizacja zarówno współrzędnych wejściowych (ξ, η), jak i wyjść sieci za pomocą skal σ_0 i u_0 okazała się niezbędna dla uzyskania zbieżności. Przyjęte skale nie muszą wynikać z rozwiązania analitycznego — w praktyce pomocne jest wstępne przeprowadzenie obliczeń dla parametrów zadania zbliżonych do jedności (parametry materiałowe, wymiary geometrii), co pozwala zweryfikować poprawność sformułowania modelu PINN niezależnie od problemów ze skalowaniem. Jeśli trening zbiega dla takich parametrów, potwierdza to poprawność implementacji, a ewentualne trudności ze zbieżnością dla docelowych wartości wskazują na konieczność korekty współczynników normalizujących. Rekomenduje się również zdefiniowanie metryki testowej opartej na rozwiązaniu referencyjnym (analitycznym lub MES), która pozwala na bieżąco weryfikować, czy trening zmierza w oczekiwanym kierunku. Brak spadku tej metryki w pierwszych tysiącach iteracji sygnalizuje błędnie dobrane współczynniki normalizujące, architekturę sieci lub hiperparametry.

Spośród czterech badanych wariantów zadania tarczowego najkorzystniejszy stosunek dokładności do kosztu obliczeniowego uzyskano dla wariantu W2 (FNN, sformułowanie mieszane, miękkie warunki brzegowe), który osiągnął najniższy błąd ϵ_{ℓ_2} w najkrótszym łącznym czasie spośród wszystkich wariantów. Wariant W1 (sformułowanie w przemieszczeniach) wymagał znacznie dłuższego treningu, lecz jego zaletą jest fizyczna spójność — naprężenia wynikają bezpośrednio z pól przemieszczeń przez różniczkowanie i związek konstytutywny, co eliminuje ryzyko niefizycznych rozbieżności między polami. W zastosowaniach, w których priorytetem jest wewnętrzna zgodność rozwiązania z równaniami mechaniki, wariant ten pozostaje atrakcyjnym wyborem pomimo wyższego kosztu.

Architektura PFNN (wariant W3) nie przyniosła wymiernej poprawy dokładności względem standardowej sieci FNN o porównywalnej liczbie parametrów, a jednocześnie wydłużyła czas obliczeń. W rozpatrywanym problemie składowe przemieszczeń i naprężeń są ściśle sprzężone przez równania równowagi oraz związek konstytutywny, co podważa zasadność ich niezależnej parametryzacji w odrębnych gałęziach sieci. Warto odnotować, że architektura PFNN jest zastosowana w oficjalnym przykładzie biblioteki DeepXDE dla zadania sprężystości płyty [41], jednak przeprowadzone eksperymenty nie potwierdzają jej przewagi w tym typie zagadnień.

Wariant W4 (twarde warunki brzegowe) eliminuje składnik L_{BC} z funkcji kosztu dzięki algebraicznym mnożnikom zerującym, co upraszcza strukturę problemu optymalizacyjnego. Uzyskany błąd okazał się istotnie wyższy niż w wariantach W1–W3, jednak różnica ta wynika przede wszystkim ze zmiany rozwiązania referencyjnego — wariant W4 jako jedyny jest porównywany z MES, a nie z rozwiązaniem TS.

Dwuetapowy trening (Adam + L-BFGS) okazał się skuteczną strategią we wszystkich wariantach, przy czym eksperymenty pomocnicze dla W2 wykazały, że sam optymalizator L-BFGS jest w stanie osiągnąć satysfakcjonującą dokładność w czasie kilkudziesięciu sekend bez jakiejkolwiek fazy Adama. Tak krótki czas obliczeń sugeruje, że analogiczne eksperymenty byłyby wykonalne na konsumenckich kartach graficznych, co otwiera możliwość szerszego stosowania metody PINN. Jednocześnie obserwacja, że skrócona faza Adama (30×10^3 iteracji) pogorszyła końcowy wynik względem wariantu bez Adama, wskazuje na wrażliwość procesu optymalizacji na punkt startowy fazy L-BFGS — częściowy trening Adamem może prowadzić sieć do obszaru przestrzeni parametrów, z którego L-BFGS trudniej osiąga globalne minimum.

Eksperymenty z wariantem W4 potwierdziły również istotny wpływ liczby punktów residuum na zbieżność rozwiązania. Przy rzadkim próbkowaniu domeny trening nie osiągał zbieżności — dopiero zwiększenie liczby punktów residuum do 40 000, wartości porównywalnej z liczbą elementów skończonych w modelu referencyjnym MES, pozwoliło uzyskać rozwiązanie. Analogia z gęstością dyskretyzacji MES nie jest przypadkowa — w obu podejściach dokładność zależy od liczby punktów, w których wymusza się spełnienie równań.

Uzyskane wyniki potwierdzają skuteczność metody PINN w rozwiązywaniu zadań liniowej teorii sprężystości, jednak w tej klasie problemów metoda nie oferuje przewagi nad MES pod względem dokładności ani czasu obliczeń. Potencjał PINN ujawnia się w zagadnieniach, w których klasyczne podejście wymaga wielokrotnego rozwiązywania układu równań — w szczególności w problemach odwrotnych oraz optymalizacji parametrów materiałowych. Zastosowanie PINN do takich zadań, obejmujące materiał hipersprężysty i identyfikację jego parametrów konstytutywnych, jest przedmiotem kolejnego rozdziału.

8. Identyfikacja relacji konstytutywnych materiałów hipersprężystych z wykorzystaniem platformy PINNACLE[2] na podstawie syntetycznych pól przemieszczeń

8.1. Wprowadzenie

W poprzednich rozdziałach parametry modeli hipersprężystości wyznaczano klasycznie — dopasowując krzywą naprężenie–wydłużenie do danych z testów Treloara [33] (rozciąganie jednoosiowe, dwuosiowe, ścinanie czyste). W praktyce laboratoryjnej najłatwiej dostępna jest próba jednoosiowego rozciągania. Wykonanie drugiego niezależnego testu (np. równomiernego rozciągania dwuosiowego lub czystego ścinania) wymaga specjalistycznej aparatury i jest znacznie trudniejsze do zrealizowania. Tymczasem, jak wykazano w [79], [139], [140], parametry wyznaczone wyłącznie z jednego testu nie przenoszą się poprawnie na inne stany deformacji — model skalibrowany na próbie jednoosiowej może dawać istotne błędy zarówno w próbie dwuosiowej czy w czystym ścinaniu, jak i w złożonym, niejednorodnym stanie deformacji. Wynika to z natury problemu: nieliniowe relacje konstytutywne hipersprężystości opisują zachowanie materiału w przestrzeni wieloosiowych stanów odkształceń, a pojedynczy test eksploruje jedynie wąski wycinek tej przestrzeni. W literaturze postuluje się więc wyznaczanie parametrów na podstawie co najmniej dwóch niezależnych testów [66], [67]. Atrakcyjną alternatywą jest wykonanie *jednego* testu o niejednorodnym stanie deformacji i rejestracji pól przemieszczeń za pomocą cyfrowej korelacji obrazu (DIC) — taki pomiar dostarcza jednocześnie informacji o odpowiedzi materiału w różnych stanach deformacji.

Cyfrowa korelacja obrazu (*Digital Image Correlation*, DIC)[141] jest bezkontaktową techniką optyczną umożliwiającą pomiar pól przemieszczeń i odkształceń na powierzchni obciążanej próbki. Metoda polega na śledzeniu przesunięć małych podobszarów (fasetek) nieregularnego wzorca nałożonego na badaną powierzchnię, zarejestrowanego przez układ kamer w kolejnych krokach obciążenia. Porównanie obrazu zdeformowanego z obrazem referencyjnym pozwala wyznaczyć mapę przemieszczeń i odkształceń z dokładnością

[2] PINNACLE – *Physics-Informed Neural Network with Adaptive Constitutive Law Engine.*

rzędu setnych części piksela [84], [87], [142]. Przykładowe stanowisko pomiarowe z systemem ARAMIS 2M wykorzystane przez autora w badaniach łożysk elastomerowych [140] przedstawiono na rys. 8.1. Co istotne, próg dostępności DIC obniża się: obok systemów komercyjnych dostępne są otwartoźródłowe implementacje, takie jak μDIC [85] czy py2DIC [86], wymagające jedynie standardowych kamer. W niniejszej pracy pomiary DIC nie są wykonywane fizycznie — ich rolę pełnią syntetyczne dane pola przemieszczeń wygenerowane z rozwiązania MES, co pozwala porównać zidentyfikowane parametry z ich dokładnie znanymi wartościami wejściowymi.

(a) Stanowisko pomiarowe z kamerami systemu ARAMIS 2M

(b) Wzorzec naniesiony na powierzchnię próbki

Rysunek 8.1. Pomiar z wykorzystaniem cyfrowej korelacji obrazu [140]: (a) stanowisko pomiarowe, (b) wzorzec na próbce

Alternatywą pozwalającą wzbogacić bazę doświadczalną jest wykorzystanie testów o niejednorodnym stanie odkształceń w połączeniu z analizą odwrotną. W podejściu FEMU (*Finite Element Model Updating*) [143] parametry konstytutywne optymalizuje się iteracyjnie, minimalizując rozbieżność między odpowiedzią modelu MES a danymi doświadczalnymi — siłami globalnymi lub polami przemieszczeń z DIC. Takie podejście zastosowano m.in. w pracy [140], gdzie parametry modeli neo-Hooke'a i Yeoha [16] wyznaczono z próby jednoosiowego rozciągania, a następnie uzupełniono bazę identyfikacyjną o test ściskania łożyska elastomerowego (stan niejednorodny), przeszukując przestrzeń

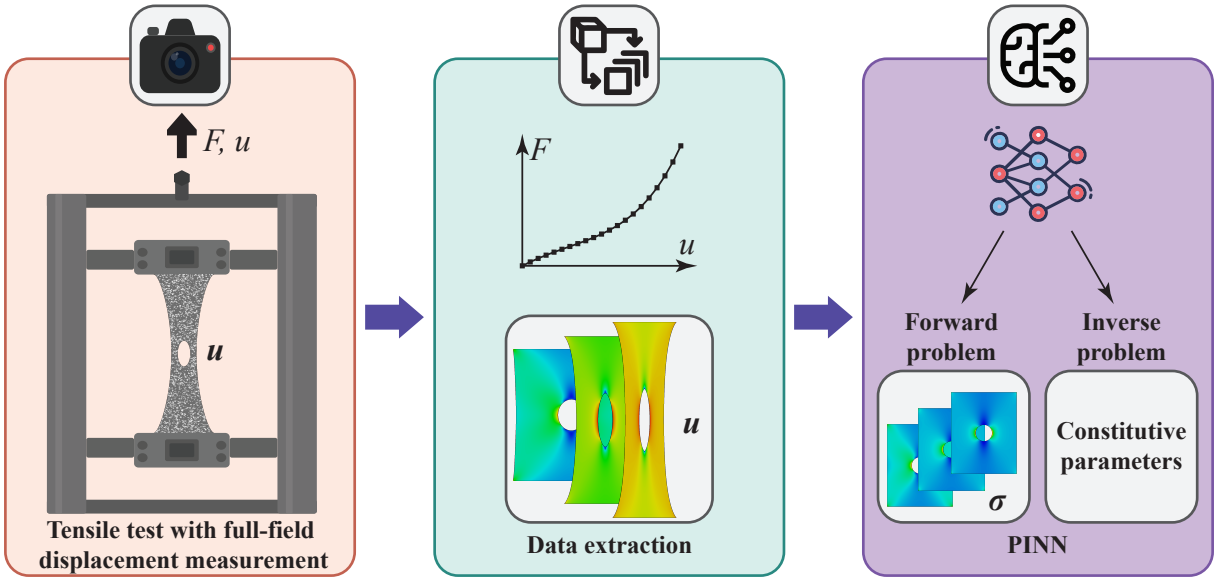
parametrów siatkowo z MES w pętli optymalizacyjnej. Kwestię doboru algorytmu optymalizacji w procedurze FEMU–DIC dla modeli sprężysto-plastycznych badano szczegółowo w [144], wskazując algorytm Powella jako najskuteczniejszy spośród metod lokalnych — przy czym procedura wymagała rzędu tysięcy ewaluacji zadań MES.

Fizycznie informowane sieci neuronowe (*Physics-Informed Neural Networks*, PINN) [2] stanowią podejście bezsiatkowe, w którym pole przemieszczeń i parametry materiałowe mogą być identyfikowane jednocześnie w ramach jednego procesu optymalizacji (treningu sieci). Nie eliminuje to kosztu obliczeniowego — trening PINN może być porównywalnie lub nawet bardziej czasochłonny niż pojedyncze uruchomienie FEMU — lecz zmienia jego charakter. Po pierwsze, PINN opiera się bezpośrednio na równaniach równowagi i nie wymaga generowania siatki MES ani agregacji macierzy sztywności. Po drugie, parametry konstytutywne $\boldsymbol{\theta}_{\text{const}}$ są optymalizowane gradientowo wraz z wagami sieci dzięki automatycznemu różniczkowaniu [42], co umożliwia jednoczesną identyfikację wielu parametrów bez potrzeby wielokrotnego uruchamiania zewnętrznego programu obliczeniowego (solvera).

8.2. Wykorzystanie sieci neuronowej informowanej fizyką PINN do identyfikacji parametrów materiałowych hipersprężystości

Schemat zbierania i przetwarzania danych w proponowanym algorytmie przedstawiono na rys. 8.2. W docelowym zastosowaniu próbka o niezidentyfikowanych parametrach materiałowych, zawierająca wewnętrzne niejednorodności geometryczne, poddawana byłaby jednoosiowemu rozciąganiu w warunkach płaskiego stanu naprężenia (PSN). Powierzchnia próbki powinna być pokryta wzorcem (ang. *speckle*), przykładowo jak na rys. 8.1b, umożliwiającym rejestrację przemieszczeń metodą cyfrowej korelacji obrazu (DIC), natomiast czujnik siły powinien rejestrować historię obciążenia w trakcie deformacji. W ten sposób uzyskuje się wieloźródłowy zbiór danych pomiarowych, obejmujący zarówno pole przemieszczeń, jak i historię siły reakcji. W niniejszej pracy dane pomiarowe zastąpiono danymi syntetycznymi generowanymi numerycznie metodą elementów skończonych (MES), co pozwala na weryfikację algorytmu we w pełni kontrolowanych warunkach, przy dokładnej znajomości parametrów materiałowych użytych do wygenerowania danych. Uzyskane w ten sposób pola przemieszczeń oraz historia siły reakcji stanowią dane wejściowe do procedury uczenia sieci PINN, której celem jest identyfikacja parametrów konstytutywnych.

Opis architektury sieci PINN przedstawiono w podrozdziale 8.3, natomiast zagadnienia brzegowego oraz parametry założonych modeli materiałowych — w podrozdziale 8.4.1.

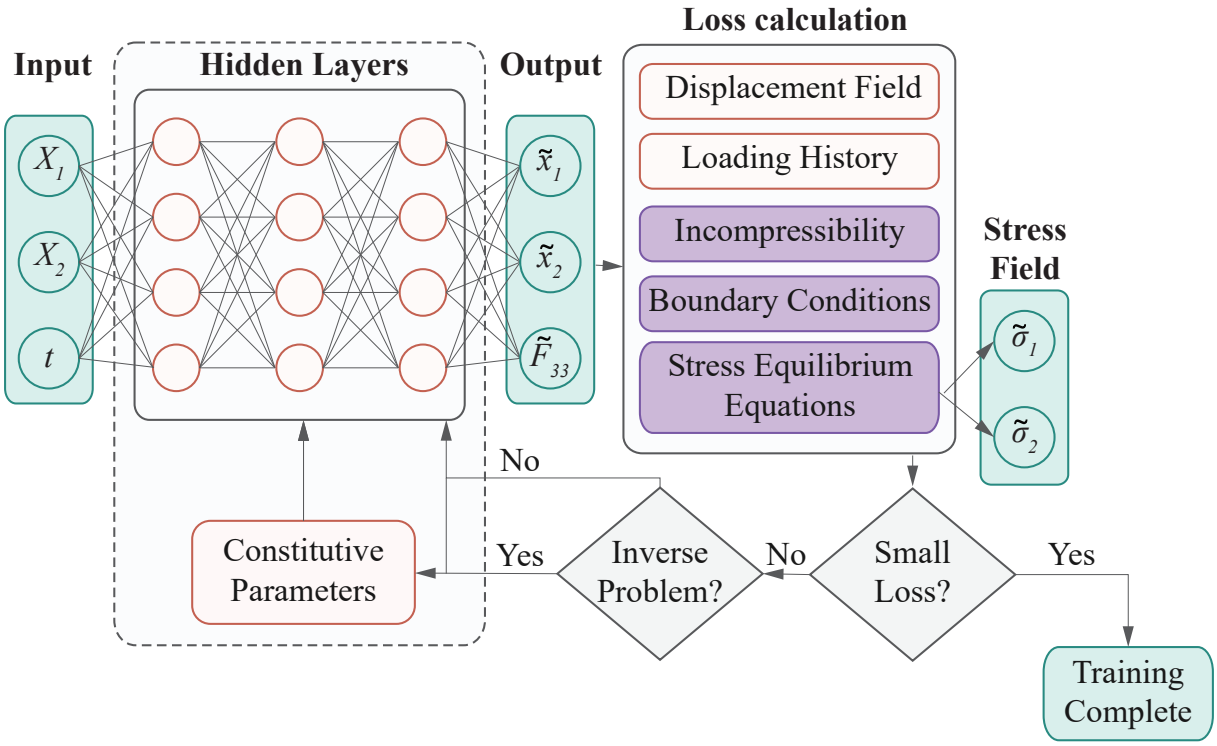


Rysunek 8.2. Schemat procedury wykorzystującej sieć PINN do analizy zadania brzegowego hipersprężystości. Procedura obejmuje trzy etapy:(1) przeprowadzenie eksperymentu rozciągania próbki z niejednorodnością przy jednoczesnej rejestracji globalnej siły F i u oraz pola przemieszczeń $\mathbf{u}$, (2) zebranie i przygotowanie danych, (3) wyznaczenie pola naprężeń $\boldsymbol{\sigma}$ w przypadku problemu prostego ze zdefiniowanymi parametrami konstytutywnymi lub wyznaczenie parametrów konstytutywnych hipersprężystości w przypadku problemu odwrotnego [145]

8.3. Architektura sieci neuronowej informowanej fizyką PINN do identyfikacji parametrów materiałowych hipersprężystości

W dalszych rozważaniach przyjęto, że indeks 1 odpowiada kierunkowi obciążenia, indeks 2 – kierunkowi poprzecznemu w płaszczyźnie, a indeks 3 – kierunkowi prostopadłemu do płaszczyzny deformacji. Wielkości aproksymowane przez sieć neuronową oznaczono tyldą $(\tilde{\cdot})$, wartości zadane na brzegu – stopniem $(\cdot)^{\circ}$, natomiast dane pomiarowe oznaczono gwiazdką $(\cdot)^{*}$. Choć w niniejszej pracy dane pomiarowe pochodzą z symulacji MES, konwencja ta odzwierciedla docelową implementację metody, w której źródłem danych będą pomiary pełnopolowe wykonane techniką cyfrowej korelacji obrazu. Nieznane parametry materiałowe modelu konstytutywnego zbieramy w wektor $\boldsymbol{\theta}_{\text{const}}$. Ich identyfikacja metodą PINN składa się z czterech kroków.

Na rysunku 8.3 przedstawiono schemat architektury sieci PINN w przypadku zadania prostego (*forward problem*) jak i odwrotnego (*inverse problem*). Zadanie proste posłuży nam do weryfikacji architektury sieci, ale w ogólności pozwala wyznaczać ciągłe pole



Rysunek 8.3. Schemat przepływu danych przez sieć neuronową PINN w zadaniu identyfikacji parametrów materiałowych i/lub wyznaczania pól naprężeń [145]

naprężeń dla znanych parametrów konstytutywnych. W fazie propagacji w przód (*forward pass*) sieć przyjmuje na wejściu położenie próbki w konfiguracji materialnej oraz krok przyrostu obciążenie t z czego przewiduje położenie próbki w konfiguracji aktualnej oraz składową gradientu deformacji $\tilde{F}_{33}$. Na ich podstawie obliczana jest wartość funkcji kosztu jako średnia ważona błędów spełnienia następujących warunków: zgodności pomierzonych pól przemieszczeń oraz globalnej siły, zachowania nieściśliwości, spełnienia warunków brzegowych i równań równowagi w naprężeniach. Następnie gradienty funkcji kosztu są propagowane wstecz, a wagi sieci aktualizowane metodą gradientową. W przypadku zadania odwrotnego wektor parametrów konstytutywnych $\boldsymbol{\theta}_{\text{const}}$ traktowany jest jako dodatkowy zbiór zmiennych optymalizacyjnych i tak jak wagi sieci podlega aktualizacji w każdej iteracji propagacji wstecznej. Umożliwia to jednoczesną identyfikację parametrów materiałowych jak i wyznaczenie pola naprężeń.

8.3.1. Aproksymacja kinematyki siecią neuronową

Zaczynamy od tego, że sieć neuronowa przybliża pole deformacji. Na wejściu podajemy współrzędne materialne punktu $\mathbf{X} = (X_1, X_2)$ w konfiguracji początkowej B_0, a sieć zwraca jego położenie w konfiguracji aktualnej $\tilde{\mathbf{x}} = (\tilde{x}_1, \tilde{x}_2)$ oraz składową gradientu deformacji

poza płaszczyzną $\tilde{F}_{33}$. Zapisujemy to jako:

$$(\tilde{\mathbf{x}},\, \tilde{F}_{33}) = \mathcal{N}_u(\mathbf{X},\, t;\, \boldsymbol{\theta}_{\mathrm{NN}},\, \boldsymbol{\theta}_{\mathrm{const}}), \tag{8.1}$$

gdzie t to bezwymiarowa zmienna parametryzująca quasi-statyczny proces obciążenia próbki. Tylda nad symbolem oznacza, że mamy do czynienia z przybliżeniem z sieci neuronowej, a nie z wartością dokładną. Wektor $\boldsymbol{\theta}_{\mathrm{NN}}$ zawiera wszystkie trenowane parametry sieci neuronowej (wagi i wyrazy wolne).

Rozciąganie próbki jest procesem quasi-statycznym. Zamiast czasu fizycznego, ścieżkę obciążenia parametryzuje się zmienną $t \in [0; 1]$, która porządkuje kolejne konfiguracje ciała. Wydłużenia próbki zmienia się liniowo:

$$\lambda(t) \;=\; 1 + (\lambda_{\max} - 1)\, t. \tag{8.2}$$

Przy $t = 0$ próbka znajduje się w konfiguracji odniesienia ($\lambda = 1$, brak deformacji), natomiast przy $t = 1$ osiąga zadany stopień wydłużenia $\lambda_{\max}$. Przemieszczenie punktu materialnego wyznacza się klasycznie jako różnicę położeń:

$$\tilde{\mathbf{u}}(\mathbf{X},\, t) = \tilde{\mathbf{x}} - \mathbf{X}. \tag{8.3}$$

Należy podkreślić, że t **nie jest czasem fizycznym** — rozpatrywany materiał hipersprężysty nie wykazuje zależności od czasu (brak lepkości ani plastyczności). Jest to wyłącznie parametr porządkujący kolejne stany równowagi w procesie quasi-statycznego obciążania, co w kontekście implementacji PINN ma istotne konsekwencje dla sposobu konstrukcji dziedziny obliczeniowej.

8.3.2. Wbudowanie związku konstytutywnego w sieć

Wszystkie wielkości mechaniczne wyprowadzamy bezpośrednio z wyjść sieci, korzystając z automatycznego różniczkowania (autodiff). Dzięki temu gradient deformacji liczymy jako:

$$\tilde{\mathbf{F}}(\mathbf{X}, t) = \frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{X}}, \tag{8.4}$$

a tensor naprężenia Pioli-Kirchhoffa I rodzaju $\tilde{\mathbf{S}}$ dla materiału hipersprężystego ma postać:

$$\tilde{\mathbf{S}}(\mathbf{X},t) = -\tilde{p}(\mathbf{X},t)\,\tilde{\mathbf{F}}^{-T}(\mathbf{X},t) + 2\,\tilde{\mathbf{F}}(\mathbf{X},t)\,\frac{\partial \tilde{W}(\mathbf{X},t)}{\partial \tilde{\mathbf{C}}(\mathbf{X},t)}, \tag{8.5}$$

gdzie $\tilde{p}$ to ciśnienie hydrostatyczne (w płaskim stanie naprężenia wyznaczane z warunku równowagi $\tilde{S}_{33} = 0$)), $\tilde{W}$ to jednostkowa energia sprężystości (ES), a

$$\tilde{\mathbf{C}}(\mathbf{X},t) = \tilde{\mathbf{F}}^{T}(\mathbf{X},t)\,\tilde{\mathbf{F}}(\mathbf{X},t) \tag{8.6}$$

to prawy tensor deformacji Cauchy'ego–Greena.

8.3.3. Sformułowanie zagadnienia i funkcja kosztu

W omawianym zagadnieniu quasi-statycznym (bez sił bezwładnościowych), z pominiętymi siłami objętościowymi i przy założeniu nieściśliwości materiału, muszą być spełnione następujące warunki. Pierwszym z nich jest równanie równowagi w konfiguracji odniesienia (por. (5.31)):

$$\mathrm{DIV}\,\tilde{\mathbf{S}}(\mathbf{X},t) \approx \mathbf{0}. \tag{8.7}$$

Kolejnym — warunek nieściśliwości:

$$\det \tilde{\mathbf{F}}(\mathbf{X},t) \approx 1. \tag{8.8}$$

Na brzegu zadajemy warunki brzegowe – przemieszczeniowe i naprężeniowe:

$$\tilde{\mathbf{u}}(\mathbf{X},t) \approx \mathbf{u}^{\circ}(\mathbf{X},t), \tag{8.9}$$

$$\tilde{\mathbf{S}}(\mathbf{X},t)\,\mathbf{N}(\mathbf{X}) \approx \mathbf{p}^{\circ}(\mathbf{X},t), \tag{8.10}$$

gdzie $\mathbf{N}(\mathbf{X})$ to zewnętrzny wektor normalny do brzegu, $\mathbf{u}^{\circ}$ to zadane przemieszczenia, a $\mathbf{p}^{\circ}$ to zadane obciążenia powierzchniowe.

W trakcie treningu sieci wykorzystujemy również dane referencyjne pochodzące z rozwiązania uzyskanego metodą elementów skończonych (MES). W celu odwzorowania niepewności pomiarowej charakterystycznej dla danych pochodzących z cyfrowej korelacji obrazu do każdej składowej przemieszczenia wprowadzono losowy szum o amplitudzie 1%. Tak zaburzone przemieszczenia $\mathbf{u}^{*}(\mathbf{X}_{\mathrm{REF}},t)$, traktowane są jako dodatkowe warunki

zgodności – rozwiązanie sieci powinno być z nimi spójne:

$$\tilde{\mathbf{u}}(\mathbf{X}_{\mathrm{REF}}, t) \approx \mathbf{u}^*(\mathbf{X}_{\mathrm{REF}}, t). \tag{8.11}$$

Dodatkowo zakładamy, że całkowita siła powierzchniowa na brzegu obciążonym powinna zgadzać się z historią obciążenia $\mathbf{p}^*(t)$ wyznaczoną w symulacji MES:

$$\int_{\partial\Omega_t} \tilde{\mathbf{S}}(\mathbf{X}, t)\, \mathbf{N}(\mathbf{X})\, \mathrm{d}A \approx \mathbf{p}^*(t), \tag{8.12}$$

Warunek ten stabilizuje identyfikację parametrów, co jest szczególnie istotne przy dużych deformacjach, gdzie silna nieliniowość relacji konstytutywnej powoduje, że samo pole przemieszczeń nie wystarcza do jednoznacznego wyznaczenia parametrów materiałowych.

W praktyce okazuje się jednak, że bezpośrednie wyznaczanie siły z punktów kolokacyjnych leżących na obciążonym brzegu prowadzi do problemów numerycznych. Optymalizator PINN minimalizuje funkcję kosztu jednocześnie we wszystkich punktach kolokacyjnych – zarówno tych odpowiadających równaniom równowagi w dziedzinie, jak i tych wymuszających zgodność siły na brzegu. Te dwa warunki wchodzą ze sobą w konflikt: punkty kolokacyjne na obciążonym brzegu muszą jednocześnie spełniać warunek zgodności siły obciążającej i równania równowagi, podczas gdy punkty wewnętrzne odpowiadają wyłącznie za równowagę wewnętrzną. Dodatkowy warunek na brzegu utrudnia optymalizatorowi jednoczesne spełnienie równań równowagi w całej dziedzinie. Skutkiem jest sztuczne przewężenie próbki.

Żeby tego uniknąć, proponuje się estymację siły przez uśrednianie naprężeń po całej dziedzinie metodą Monte Carlo. Z równań równowagi wynika, że całka naprężeń po dowolnym przekroju próbki daje tę samą siłę wypadkową – można ją więc wyznaczyć globalnie, losując punkty z wnętrza dziedziny. Obciążenie jednostkowe w kierunku X_1 estymujemy jako:

$$p_1(t) \approx \frac{1}{|\Omega_p|} \sum_{\mathbf{X}_p \in \Omega_p} \tilde{p}_1(\mathbf{X}_p, t), \tag{8.13}$$

gdzie Ω_p to zbiór punktów wylosowanych z wnętrza dziedziny, z pominięciem obszarów przy brzegach i ewentualnych lokalnych niejednorodności. Tak sformułowany warunek nie faworyzuje żadnego fragmentu dziedziny i nie wchodzi w konflikt z równaniami równowagi, dzięki czemu próbka zachowuje poprawny kształt.

Funkcja kosztu, którą minimalizujemy podczas uczenia, składa się z pięciu członów odpowiadających powyższym warunkom [136]:

$$L = \alpha_1 L_{\mathrm{PDE}} + \alpha_2 L_{\mathrm{INC}} + \alpha_3 L_{\mathrm{BC}} + \alpha_4 L_{\mathrm{LOAD}} + \alpha_5 L_{\mathrm{REF}}, \tag{8.14}$$

gdzie współczynniki α_i regulują wagę poszczególnych warunków. Każdy człon ma postać średniego błędu kwadratowego po odpowiednim zbiorze punktów kolokacyjnych. Poszczególne człony zestawiono w tab. 8.1.

Tabela 8.1. Człony funkcji straty odpowiadające poszczególnym warunkom zagadnienia

Warunek	Równanie
Równowaga	$L_{\mathrm{PDE}} = \frac{1}{N_{\mathrm{PDE}}} \sum_{i=1}^{N_{\mathrm{PDE}}} \left(\mathrm{Div}\, \tilde{\mathbf{S}}(\mathbf{X}_i, t)\right)^2$
Nieściśliwość	$L_{\mathrm{INC}} = \frac{1}{N_{\mathrm{PDE}}} \sum_{i=1}^{N_{\mathrm{PDE}}} \left(\det \tilde{\mathbf{F}}(\mathbf{X}_i, t) - 1\right)^2$
Warunki brzegowe	$L_{\mathrm{BC}} = \frac{1}{N_{\mathrm{BC}_u}} \sum_{i=1}^{N_{\mathrm{BC}_u}} \lvert\tilde{\mathbf{u}}(\mathbf{X}_i,\, t) - \mathbf{u}^{\circ}(\mathbf{X}_i,\, t)\rvert^2 + \frac{1}{N_{\mathrm{BC}_p}} \sum_{i=1}^{N_{\mathrm{BC}_p}} \left\lvert\tilde{\mathbf{S}}(\mathbf{X}_i,\, t)\, \mathbf{N}(\mathbf{X}_i) - \mathbf{t}_0^{\circ}(\mathbf{X}_i,\, t)\right\rvert^2$
Historia obciążenia	$L_{\mathrm{LOAD}} = \frac{1}{N_{\mathrm{LOAD}}} \sum_{i=1}^{N_{\mathrm{LOAD}}} \left(p_1(t_i) - p_1^*(t_i)\right)^2$
Dane referencyjne (MES)	$L_{\mathrm{REF}} = \frac{1}{N_{\mathrm{REF}}} \sum_{i=1}^{N_{\mathrm{REF}}} \left(\tilde{\mathbf{u}}(\mathbf{X}_{\mathrm{REF},i}, t) - \mathbf{u}^*(\mathbf{X}_{\mathrm{REF},i}, t)\right)^2$

Symbole N z odpowiednimi indeksami oznaczają liczby punktów kolokacyjnych w poszczególnych zbiorach.

8.3.4. Optymalizacja

Na koniec minimalizujemy całkowitą funkcję straty jednocześnie po parametrach sieci $\boldsymbol{\theta}_{\mathrm{NN}}$ i po nieznanych stałych materiałowych $\boldsymbol{\theta}_{\mathrm{const}}$:

$$(\hat{\boldsymbol{\theta}}_{\mathrm{NN}},\, \hat{\boldsymbol{\theta}}_{\mathrm{const}}) = \arg \min_{\boldsymbol{\theta}_{\mathrm{NN}},\, \boldsymbol{\theta}_{\mathrm{const}}} L(\boldsymbol{\theta}_{\mathrm{NN}},\, \boldsymbol{\theta}_{\mathrm{const}}), \tag{8.15}$$

gdzie „daszek" nad symbolem oznacza wartości wyznaczone w wyniku trenowania.

8.4. Rozwiązywanie zadań

Opisane podejście zweryfikowano na konkretnym studium przypadku. Analizowana tarcza ma kształt prostokąta o wymiarach $2l \times 2c \times b$ (długość × szerokość × grubość), z narożami w punktach $(\pm l, \pm c)$ oraz okrągłym otworem o promieniu r w centrum. Próbka poddawana jest jednoosiowemu rozciąganiu w warunkach stanu płaskiego naprężenia. Lewa krawędź ($X_1 = -l$) jest utwierdzona w obu kierunkach, natomiast na prawej krawędzi ($X_1 = l$) zadano przemieszczenie w kierunku obciążenia $u_1^\circ = (\Lambda - 1) \cdot 2l$ przy zerowym przemieszczeniu w kierunku poprzecznym $u_2^\circ = 0$. Krawędzie górna ($X_2 = c$) i dolna ($X_2 = -c$) pozostają swobodne, tj. wektor naprężenia na tych brzegach znika: $\mathbf{S}\,\mathbf{N} = \mathbf{0}$ (równoważnie, w konfiguracji aktualnej: $\boldsymbol{\sigma}\,\mathbf{n} = \mathbf{0}$). Analogiczny warunek obowiązuje na brzegu otworu. Obecność otworu powoduje koncentrację naprężeń i niejednorodność pola deformacji, co jest kluczowe z punktu widzenia identyfikacji parametrów materiałowych — różne obszary ciała doświadczają różnych stanów odkształcenia, umożliwiając jednoczesne wyznaczanie wielu parametrów konstytutywnych z jednego eksperymentu. Geometrię tarczy wraz z warunkami brzegowymi i siatką elementów skończonych przedstawiono na rys. 8.4, dane geometryczne zaś zestawiono w tab. 8.2.

Tabela 8.2. Dane geometryczne próbki

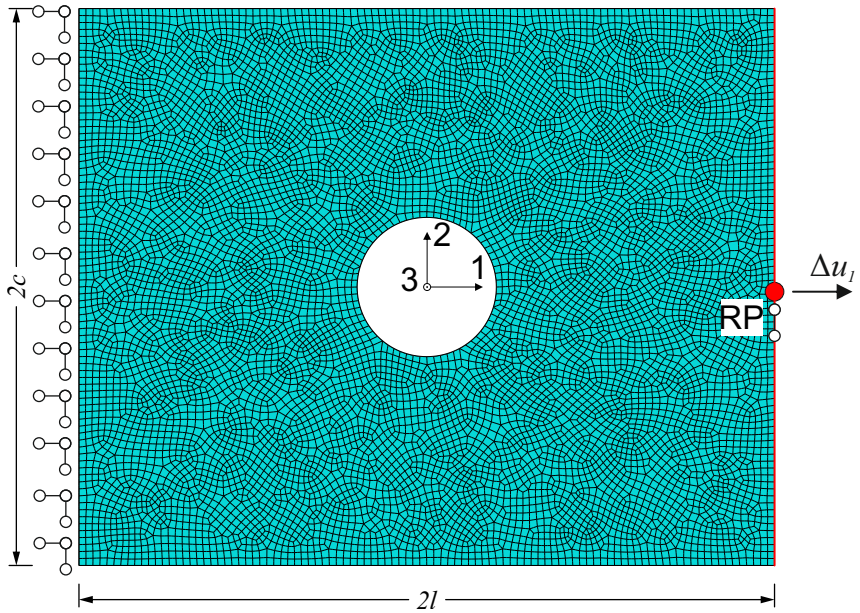
Parametr	l	c	b	r	$\lambda_{\max}$
Wartość	50 [mm]	40 [mm]	2 [mm]	10 [mm]	3.0

Model konstytutywny

Materiał przyjmuje się jako nieściśliwy. Jako model demonstracyjny wybrano model Yeoha, będący jednym z najpopularniejszych uogólnień modelu neo-Hooke'a (NH), dobrze opisujący nieliniową odpowiedź materiału przy umiarkowanych i dużych deformacji. Funkcja energii sprężystej w modelu Yeoha zależy wyłącznie od pierwszego niezmiennika $\bar{I}_1 = \mathrm{tr}(\bar{\mathbf{C}})$ prawego tensora deformacji Cauchy'ego–Greena i ma postać:

$$W(\bar{I}_1) = C_{10}(\bar{I}_1 - 3) + C_{20}(\bar{I}_1 - 3)^2 + C_{30}(\bar{I}_1 - 3)^3, \tag{8.16}$$

gdzie C_{10}, C_{20}, C_{30} to stałe materiałowe. Model ten stanowi szczególny przypadek wielomianu Rivlina przy $N = 3$, z funkcją zależną wyłącznie od I_1. Wektor parametrów



Rysunek 8.4. Geometria prostokątnej próbki z otworem z zaznaczonymi warunkami brzegowymi i siatką elementów skończonych

konstytutywnych wchodzący do (8.1) ma zatem postać:

$$\boldsymbol{\theta}_{\text{const}} = (C_{10},\, C_{20},\, C_{30}). \tag{8.17}$$

W pierwszym zadaniu wartości tych parametrów będą znane (tab. 8.3), co pozwoli zweryfikować poprawność całego schematu obliczeniowego. W zadaniu drugim parametry te będą przedmiotem identyfikacji.

Tabela 8.3. Wartości referencyjne parametrów modelu Yeoha

Parametr	Wartość	Jednostka
C_{10}	0.5	MPa
C_{20}	0.03	MPa
C_{30}	0.005	MPa

8.4.1. Zadanie referencyjne MES

W celu wyznaczenia syntetycznych danych referencyjnych przeprowadzono numeryczną symulację rozciągania tarczy z centralnym otworem. Analizę wykonano w płaskim stanie naprężenia z uwzględnieniem dużych deformacji (opcja `NLGEOM`).

Model materiału opisano modelem Yeoha z parametrami zestawionymi w tab. 8.3. Geometria tarczy odpowiada opisowi z poprzedniego podrozdziału — por. tab. 8.2 i rys. 8.4,

na którym zaznaczono warunki brzegowe, siatkę elementów skończonych a także węzeł referencyjny. Lewa krawędź ($X_1 = -l$) została utwierdzona w obu kierunkach ($u_1 = u_2 = 0$), na prawej krawędzi ($X_1 = l$) zadano przemieszczenie $u_1^\circ = (\lambda_{\max} - 1) \cdot 2l$ przy zablokowaniu drugiego kierunku. W celu uzyskania globalnej siły reakcji wprowadzono węzeł odniesienia (*Reference Point, RP*), sztywno sprzęgnięty z krawędzią przemieszczenia za pomocą opcji `MPC` (*Multi-Point Constraint*), co pozwala na jednoznaczne określenie siły wypadkowej bez sumowania reakcji w poszczególnych węzłach.

Dyskretyzację przeprowadzono ośmiowęzłowymi elementami skończonymi typu `CPS8R` (zredukowane całkowanie). Wybór ten podyktowany był wstępną analizą – elementy czterowęzłowe `CPS4R` generowały nieciągłości pola naprężeń typowe dla klepsydrowania (ang. *hourglass effect*) [146], które ustąpiły po przejściu na elementy kwadratowe z czterema punktami całkowania. Łączna liczba elementów skończonych wyniosła 9346, natomiast liczba węzłów — 28462. Zadanie rozwiązano w środowisku `ABAQUS/Standard` przy użyciu algorytmu Newtona–Raphsona w 20 równomiernych krokach obciążenia.

Dane pozyskano przy pomocy autorskiego narzędzia `odb-toolkit`[3]. Z bazy danych `.odb` dla każdej ramki obliczeniowej wyekstrahowano współrzędne węzłów siatki, ich przemieszczenia w kierunkach 1 i 2 oraz wartość siły reakcji w węźle odniesienia. W rezultacie uzyskano pole przemieszczeń $\mathbf{u}^*(\mathbf{X}, t)$ wraz z odpowiadającą mu siłą reakcji dla każdego kroku obciążenia.

8.4.2. Weryfikacja architektury sieci PINN

Przed przystąpieniem do identyfikacji parametrów materiałowych modelu konstytutywnego postanowiono przeprowadzić weryfikację poprawności architektury oraz procesu treningu sieci PINN. Celem weryfikacji jest sprawdzenie poczynionych założeń oraz dobranie hiperparametrów (głębokości i szerokości sieci, wag poszczególnych członów funkcji kosztu α_i oraz liczby punktów kolokacyjnych), zapewniających dostateczną dokładność w polach przemieszczeń i naprężeń w akceptowalnym czasie obliczeń.

Funkcję kosztu zdefiniowano zgodnie z (8.14), których współczynniki wagowe α_i zebrano w tab. 8.4.

[3] `https://github.com/Mieczmik/odb-toolkit/`

Tabela 8.4. Współczynniki α_i funkcji kosztu

Człon	**Opis**	α_i
$L_{\mathrm{PDE},1}$	Równanie równowagi (składowa 1)	1
$L_{\mathrm{PDE},2}$	Równanie równowagi (składowa 2)	1
L_{INC}	Więzy nieściśliwości	10
L_{BC}	Brzeg górny i dolny	10
L_{LOAD}	Historia obciążenia	100
$L_{\mathrm{REF},1}$	Dane MES, poł. aktualne (kierunek 1)	2000
$L_{\mathrm{REF},2}$	Dane MES, poł. aktualne (kierunek 2)	400
$L_{\mathrm{BC,hole}}$	Brzeg otworu	500

Rozmieszczenie i liczebność punktów kolokacyjnych odpowiadających poszczególnym członom funkcji kosztu zestawiono w tab. 8.5.

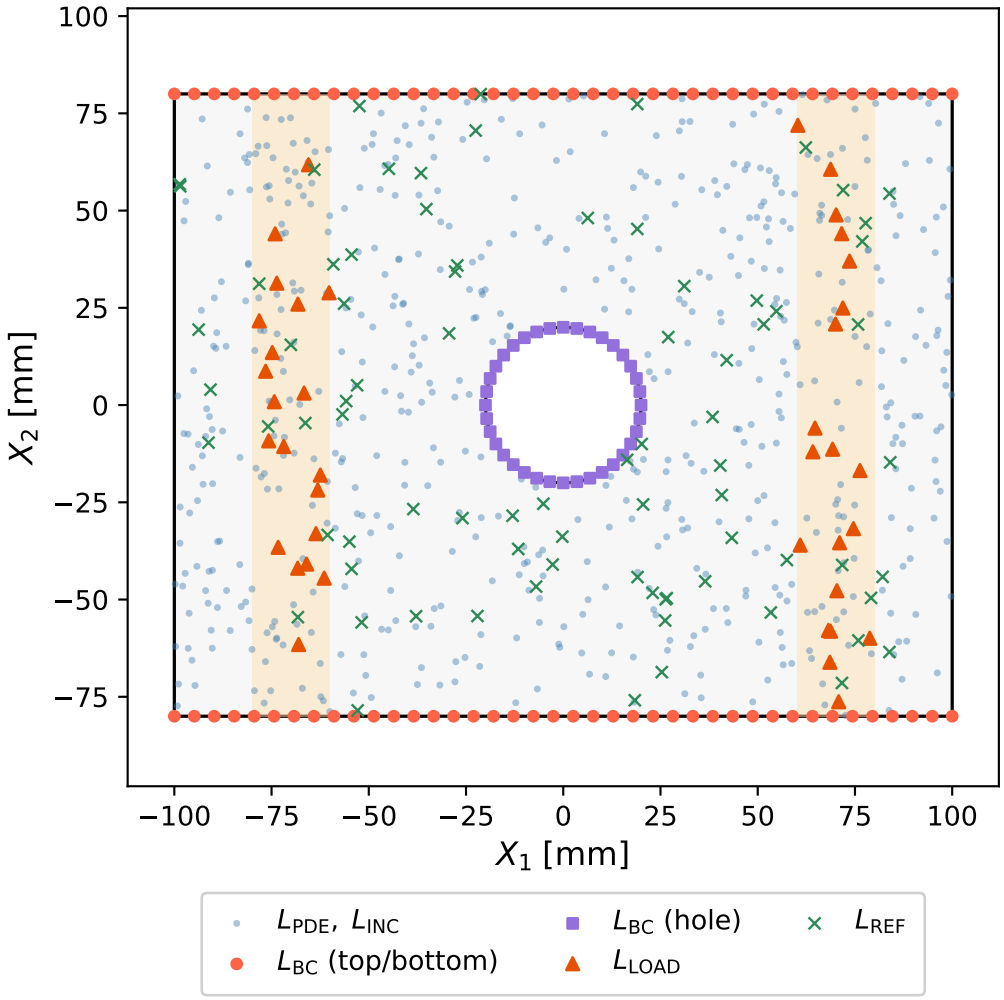
Tabela 8.5. Punkty kolokacyjne

Lokalizacja	**Składnik** L	**Liczba**	**Rozkład**
Dziedzina Ω	$L_{\mathrm{PDE}},\ L_{\mathrm{INC}}$	40 000	losowy
Brzeg górny/dolny	L_{BC} ($\mathbf{p}^\circ = \mathbf{0}$)	$2 \times 100 \times 20 = 4\,000$	równomierny
Brzeg otworu	L_{BC} ($\mathbf{p}^\circ = \mathbf{0}$)	$40 \times 20 = 800$	równomierny
Pasy $\lvert X_1\rvert \in [0.6l;\, 0.8l]$	L_{LOAD}	$400 \times 20 = 8\,000$	losowy
Węzły MES (DIC)	L_{REF}	$80 \times 20 = 1\,600$	podzbiór z szumem 1%

$n_t = 20$ oznacza liczbę kroków przyrostu obciążenia.

W dziedzinie punkty rozmieszczono losowo dla każdego przyrostu obciążenia, na krawędziach bocznych nie umieszczono punktów kolokacyjnych, ponieważ nałożono tam twarde

warunki brzegowe (hard BC)(por. s. 166). Na pozostałych brzegach dziedziny punkty rozłożono równomiernie. Punkty odpowiadające sumowaniu przyłożonego obciążenia rozmieszczono w w pasmach $|X_1| \in [0.6l;\, 0.8l]$ zgodnie z procedurą estymacji siły metodą Monte-Carlo opisaną w 8.3.3. Rozmieszczenie punktów kolokacyjnych w konfiguracji początkowej przedstawiono na rys. 8.5.

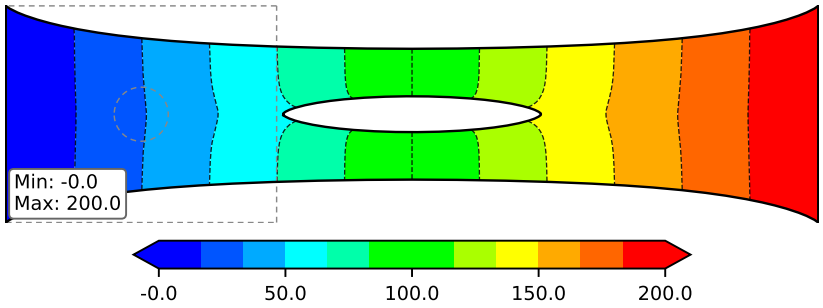


Rysunek 8.5. Rozmieszczenie punktów kolokacyjnych w konfiguracji początkowej

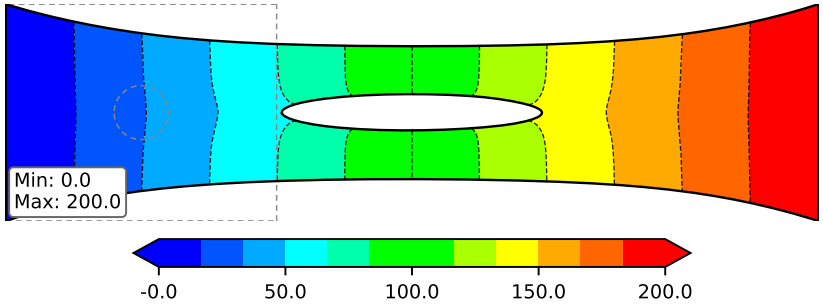
Trening przebiegał dwuetapowo. W pierwszej fazie zastosowano metodę optymalizacji Adam z krokiem uczenia $\eta = 10^{-3}$ przez 10 000 iteracji, a następnie dostrojono wagi modelu algorytmem L-BFGS-B. Procedura treningu wynika z wniosków rozdziału 7.4.3, w którym to wykazano, że wyniki uzyskane zarówno przy krótszej i dłuższej pierwszej fazie, dają zbliżone rezultaty po drugiej, a znacząco skracają czas obliczeń. Trening wynosił około 10 minut w zależności od doboru parametrów i hiperparametrów sieci.

Pola przemieszczeń.

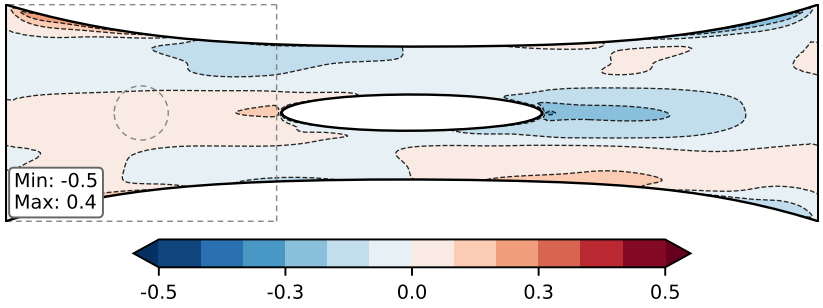
Na rys. 8.6 oraz 8.7 przedstawiono pola przemieszczeń uzyskane z sieci PINN, które porównano z rozwiązaniem MES dla końcowego kroku obciążenia.



(a) MES



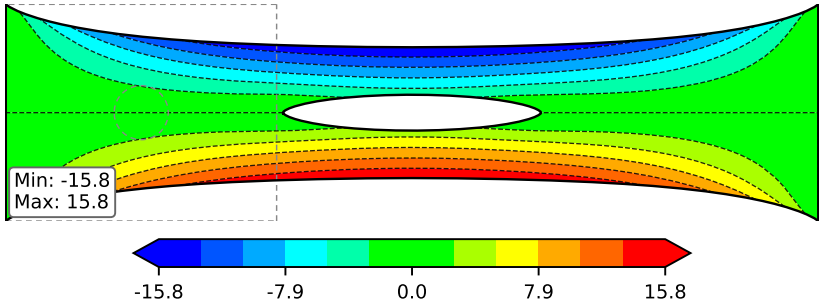
(b) PINN



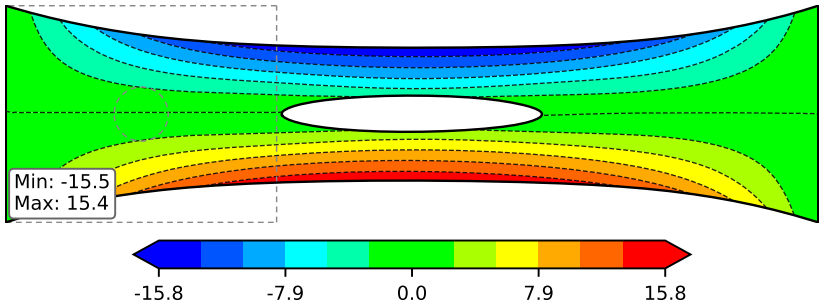
(c) $u_1^{\text{PINN}} - u_1^{\text{MES}}$

Rysunek 8.6. Pole przemieszczeń u_1[mm] w konfiguracji aktualnej: (a) rozwiązanie MES, (b) predykcja PINN, (c) błąd względny $u_1^{\text{PINN}} - u_1^{\text{MES}}$. Linią przerywaną zaznaczono konfigurację początkową.

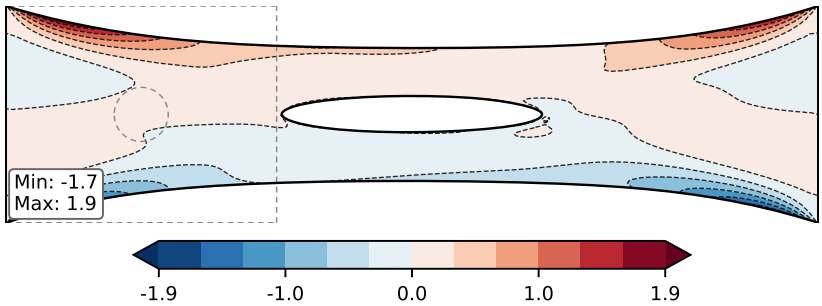
Zgodność pól u_1 jest bardzo dobra — maksymalna różnica bezwzględna nie przekracza 0.5 mm.



(a) MES



(b) PINN



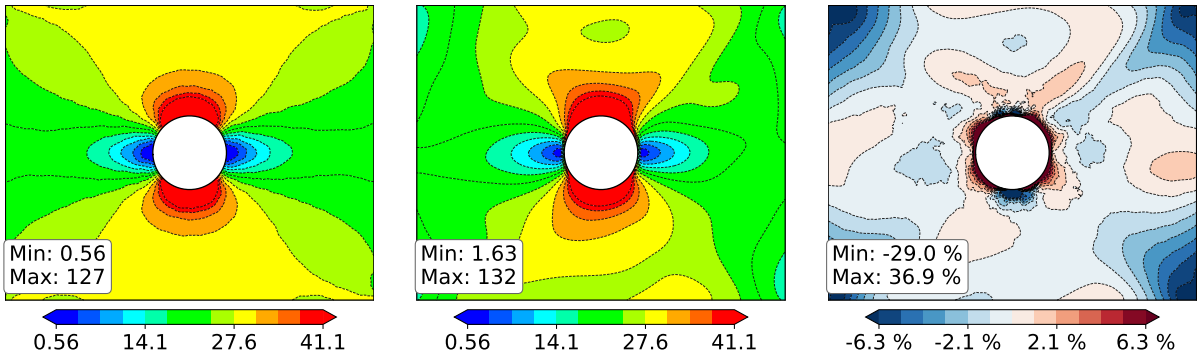
(c) $u_2^{\text{PINN}} - u_2^{\text{MES}}$

Rysunek 8.7. Pole przemieszczeń u_2[mm] w konfiguracji aktualnej: (a) rozwiązanie MES, (b) predykcja PINN, (c) błąd bezwzględny $u_2^{\text{PINN}} - u_2^{\text{MES}}$. Linią przerywaną zaznaczono konfigurację początkową.

W drugim kierunku największe rozbieżności można dostrzec na górnej i dolnej krawędzi (do 1.9 mm), zlokalizowanych w okolicach bocznych krawędzi, w których metoda PINN nie doszacowuje przewężenia. Należy zauważyć, że dla tych krawędzi warunki brzegowe zadane są w sposób miękki poprzez naprężenia i są składnikami funkcji straty podlegającymi optymalizacji w przeciwieństwie do bocznych krawędzi, na których przemieszczeniowe warunki brzegowe są spełnione z definicji.

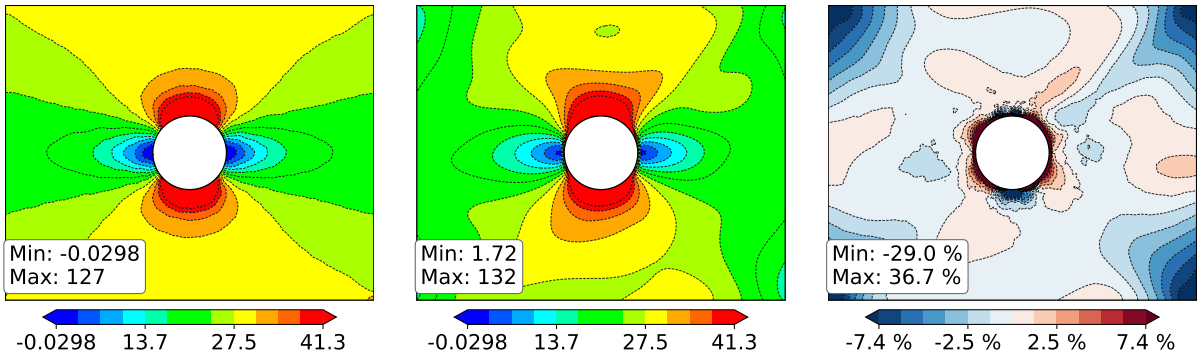
Pola naprężeń

Na rys. 8.8, 8.9 oraz 8.10 porównano naprężenia Cauchy'ego — naprężenie zredukowane Hubera–Misesa oraz naprężenia główne $\sigma_{\max}$ i $\sigma_{\min}$ uzyskane z sieci PINN z rozwiązaniem MES. Mapy błędu pokazano jako różnicę względną, $(\sigma^{\text{PINN}} - \sigma^{\text{MES}})/|\sigma^{\text{MES}}|$, wyrażoną w procentach. W celu poprawy widoczności warstwic na mapach skalę kolorystyczną map naprężeń obcięto do 99 percentyla.



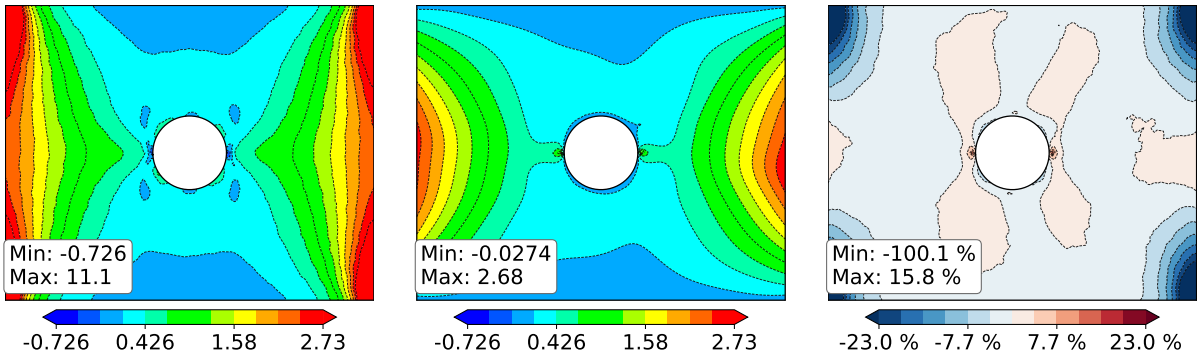
(a) MES **(b)** PINN **(c)** Błąd względny

Rysunek 8.8. Naprężenie zredukowane Hubera–Misesa σ_{HM}[MPa] w konfiguracji aktualnej: (a) rozwiązanie MES, (b) predykcja PINN, (c) błąd względny $(\sigma^{\text{PINN}} - \sigma^{\text{MES}})/|\sigma^{\text{MES}}|$



(a) MES **(b)** PINN **(c)** Błąd względny

Rysunek 8.9. Maksymalne naprężenie główne $\sigma_{\max}$[MPa] w konfiguracji aktualnej: (a) rozwiązanie MES, (b) predykcja PINN, (c) błąd względny $(\sigma^{\text{PINN}} - \sigma^{\text{MES}})/|\sigma^{\text{MES}}|$



(a) MES **(b)** PINN **(c)** Błąd względny

Rysunek 8.10. Minimalne naprężenie główne $\sigma_{\min}$[MPa] w konfiguracji aktualnej: (a) rozwiązanie MES, (b) predykcja PINN, (c) błąd względny $(\sigma^{\text{PINN}} - \sigma^{\text{MES}})/|\sigma^{\text{MES}}|$

Ekstrema naprężeń wszystkich pól uzyskane przy pomocy sieci PINN zgadzają się z wynikami uzyskanymi w MES. Ogólne charakterystyki tych pól zostały poprawnie odtworzone wraz z koncentracją naprężeń przy otworze. Największe różnice można dostrzec w bliskim otoczeniu otworu i w narożnikach próbki — szczególnie w przypadku minimalnych naprężeń głównych. Pierwsza z nich może wynikać z samej architektury sieci — funkcje aktywacji są funkcjami gładkimi i mogą mieć problem z odtworzeniem ostrych gradientów pól. Tak naprawdę ta sama interpretacja w pewnym stopniu tyczy się również narożników próbki, natomiast głównym powodem wydaje się być założenie warunków brzegowych w zadaniu referencyjnym MES, które różni się od rzeczywistego badania przy wykorzystaniu cyfrowej korelacji obrazu. W rzeczywistym badaniu próbka umieszczona jest w szczękach maszyny wytrzymałościowej albo mogłaby być zwulkanizowana ze stalowymi uchwytami, na których zachodziłyby dodatkowe efekty, takie jak tarcie, luzy, które są nieobecne w symulacji numerycznej. Z tego powodu oraz zgodnie z zasadą Saint-Venanta efekt ma charakter lokalny i jest wygaszany w pewnej odległości od brzegu. Z tego powodu porównywanie pól naprężeń w tym obszarze jest nieuzasadnione.

Mimo znaczących różnic w naprężeniach, nie wpływa to negatywnie na pole przemieszczeń. Wynika to z faktu, że naprężenia są uwzględniane w funkcji kosztu w równaniach równowagi i warunkach brzegowych tylko pośrednio, poprzez gradient tensora deformacji i związek konstytutywny. Brak jest bezpośredniego porównania z polem naprężeń uzyskanym w symulacji MES, co miało miejsce w zadaniach z rozdziału 7.4.3. Można powiedzieć, że trafiają one do analizy trochę „po drodze", a nie wprost. Takie założenie jest zamierzone,

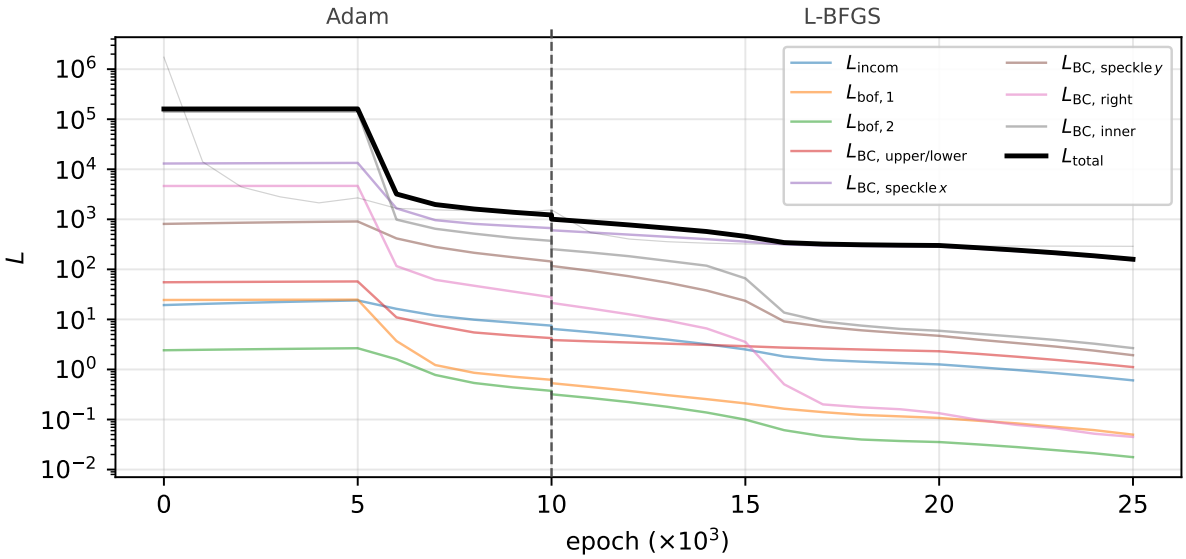
gdyż odpowiada warunkom eksperymentu przeprowadzanego docelowo przy wykorzystaniu cyfrowej korelacji obrazu (DIC).

Opis architektury i próby poprawy dokładności wyznaczania naprężeń

Funkcjami aktywacji w warstwach ukrytych był tanh, a wagi zostały zainicjalizowane metodą Glorot uniform [38]. Bazując na wnioskach płynących z 7.4.3, w trakcie dostrajania modelu podjęto szereg prób mających na celu redukcję wspomnianych różnic w naprężeniach, takich jak pogłębienie i poszerzenie architektury, zwiększenie liczby punktów kolokacyjnych w całej dziedzinie, jak i zagęszczenie ich przy otworze oraz zmniejszenie kroku uczenia w fazie optymalizacji metodą Adam. Tak naprawdę dostrajanie modelu miało charakter marginalny i w żadnym z przypadków nie zaobserwowano wyraźnej poprawy błędu ϵ_{ℓ_2}. W szczególności, jak w przypadku wcześniejszego rozwiązania, zagęszczenie punktów kolokacyjnych przy koncentracji naprężeń nie przełożyło się na poprawę wyników.

Przebieg funkcji kosztu

Na rys. 8.11 przedstawiono przebieg poszczególnych składników funkcji kosztu w trakcie treningu. Wyróżniono granicę między fazą Adama a fazą L-BFGS.

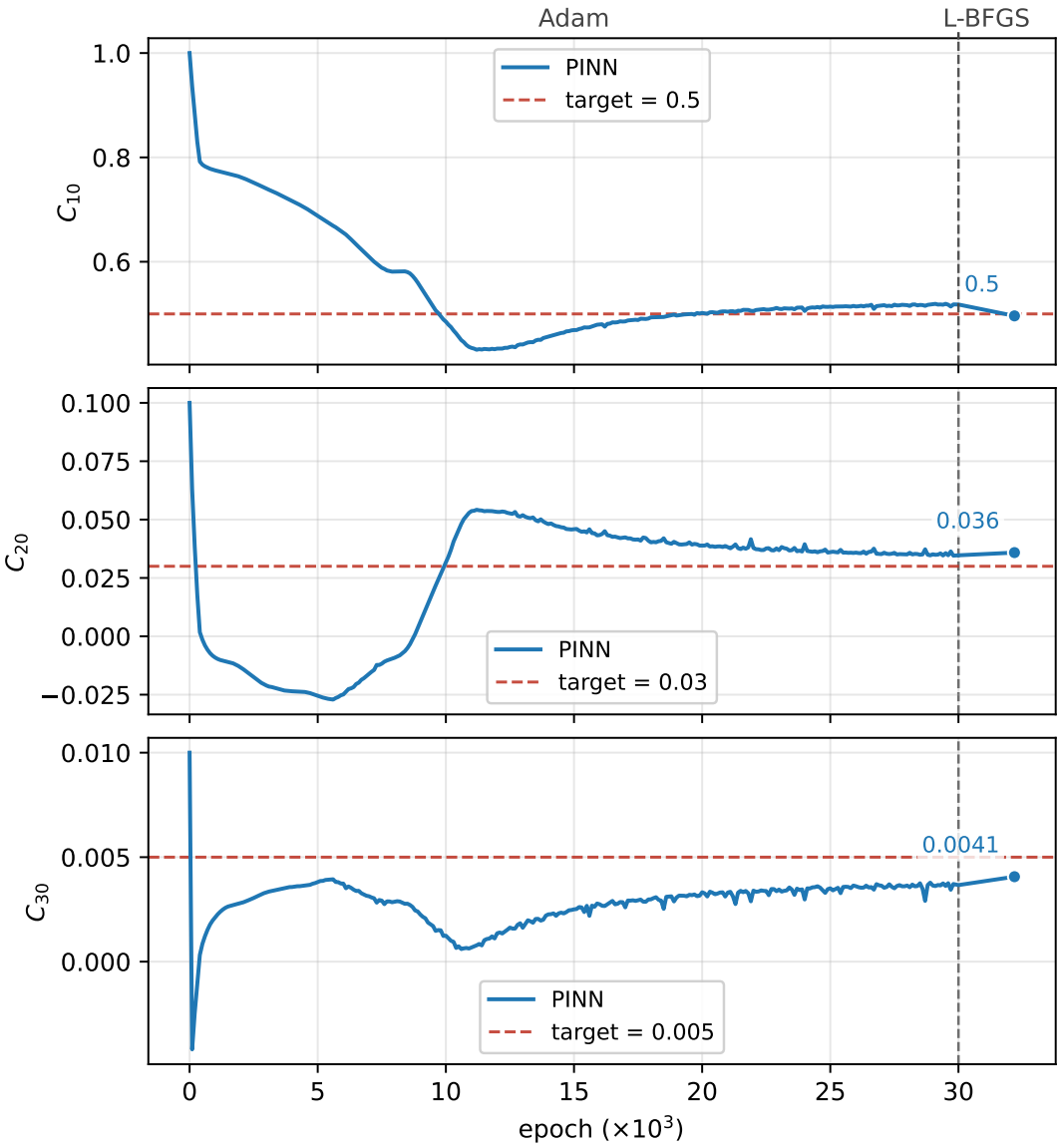


Rysunek 8.11. Wygładzony przebieg testowej funkcji straty podczas treningu sieci. Czarną linią oznaczono całkowitą wartość funkcji straty L, a kolorowymi – oznaczono odpowiednie składowe. Pionową, przerywaną linią oznaczono miejsce, w którym metoda optymalizacji zmienia się z Adam na L-BFGS.

8.4.3. Wyznaczenie parametrów materiałowych hipersprężystości

Do tego momentu rozprawy sieci PINN były wykorzystane do rozwiązywań zagadnień prostych (ang. *forward*), tzn. dla takich, w których dla zadanych z góry parametrów materiałowych sieć otrzymując na wejściu położenie punktów materialnych uczono wyznaczać pola przemieszczeń i naprężeń w rozdziale 7.4.3 czy położenie punktów w konfiguracji aktualnej 8.4.2 spełniając równania równowagi wewnętrznej i warunki brzegowe. W problemie odwrotnym, rozpatrywanym w tym podrozdziale, stałe materiałowe nie są znane *a priori*, lecz wspólnie z wagami sieci neuronowej trafiają do wspólnego problemu optymalizacyjnego. W praktyce reprezentowane są jako obiekty `dde.Variable` biblioteki DeepXDE i są kolejnymi wyrazami wektora zmiennych, które minimalizujemy w zadaniu optymalizacyjnym. Dzięki temu, z formalnego punktu widzenia, zagadnienie jest niemal identyczne jak zagadnienie proste — zmienia się jedynie skład tego wektora.

Tak jak poprzednio 8.4.2 optymalizację przeprowadzono w dwóch etapach stosując przez 30 tys. epok optymalizator Adam z krokiem uczenia $\eta = 10^{-3}$, z którego rozwiązanie dostraja się algorytmem L-BFGS. Wartości parametrów początkowych ustalono jako $\boldsymbol{\theta}^{(0)}_{\text{const}} = (C_{10},\ C_{20},\ C_{30}) = (1.0,\ 0.1,\ 0.01)$, a wartości docelowe wynoszą, zgodnie z przeprowadzonym zadaniem referencyjnym (por. tab. 8.3): $\boldsymbol{\theta}^{*}_{\text{const}} = (0.5,\ 0.03,\ 0.005)$. Przebieg zbieżności tych parametrów pokazano na rys. 8.12.



Rysunek 8.12. Przebieg identyfikacji parametrów modelu Yeoha C_{10}, C_{20}, C_{30} w trakcie treningu sieci PINN. Niebieska krzywa przedstawia bieżącą wartość identyfikowanego parametru, czerwona linia przerywana — wartość docelową. Pionowa linia zaznacza przejście z optymalizatora Adam do L-BFGS. Etykiety przy końcu krzywych podają ostateczne wartości zaokrąglone do dwóch cyfr znaczących.

We wczesnych iteracjach treningu, gdy pole przemieszczeń odbiega od rozwiązania referencyjnego, a sieć uczy się spełniać równania równowagi parametry materiałowe zmieniają się dynamicznie. Wraz ze stabilizacją pola przemieszczeń rosnącą wagę zaczynają odgrywać równania konstytutywne, co prowadzi do stopniowej zbieżności wyznaczanych parametrów materiałowych, a następnie faza L-BFGS doprecyzowuje rozwiązanie.

8.5. Opis implementacji

Kod źródłowy programu umożliwia rozwiązanie zadań prostych i odwrotnych. Umożliwia wprowadzenie dowolnej liczby otworów. Został również dostosowany do wprowadzenia karbów. Ponadto umożliwia automatyczne dostrojenie wag funkcji kosztu z wykorzystaniem algorytmu Optuna [147], [148]. Został zorganizowany w następującej strukturze:

```
config/
    config.yaml
src/
    bcs.py
    data.py
    energy_adapters.py
    integration_bc.py
    network.py
    params.py
    pde.py
    solution.py
optuna_tune.py
train.py
```

Pełny kod źródłowy zamieszczono w Załączniku 9. Poniżej krótko opisano każdy z modułów:

- `config.yaml` – plik konfiguracyjny zawierający geometrię próbki, opis obciążenia, hiperparametry sieci, parametry treningu oraz stałe materiałowe modelu,
- `train.py` – główny skrypt uruchamiający proces uczenia sieci neuronowej,
- `optuna_tune.py` – skrypt automatycznego strojenia hiperparametrów z wykorzystaniem biblioteki Optuna,
- `params.py` – wczytywanie i walidacja parametrów z pliku konfiguracyjnego,
- `network.py` – definicja architektury sieci neuronowej (PINN),
- `pde.py` – definicja residuów pochodzących z naprężeniowych równań równowagi i więzów nieściśliwości,
- `bcs.py` – definicja warunków brzegowych,

- `integration_bc.py` – rozszerzenie klasy `BC` z DeepXDE, realizujące warunki brzegowe w postaci całkowej,
- `energy_adapters.py` – moduł ACLE definiujący funkcje modeli hipersprężystości w wariantach ze stałymi parametrami — do rozwiązania problemu prostego i dynamicznie zmieniającymi się podczas treningu — do rozwiązania problemu odwrotnego,
- `data.py` – wczytywanie wyników z MES oraz generowanie dodatkowych punktów treningowych: brzegowych, do weryfikowania historii obciążenia oraz do sprawdzania zgodności z przemieszczeniami uzyskanym w eksperymencie. Możliwe jest także dogęszczenie punktów kolokacyjnych wokół otworów,
- `solution.py` – interpolacja przemieszczenia dowolnym punkcie na próbce i kroku obciążenia na podstawie wyników z MES.

8.6. Wnioski

Zastosowanie sieci neuronowych informowanych fizyką (PINN) do rozwiązywania zagadnień brzegowych hipersprężystości jest obiecujące w porównaniu z powszechnie stosowaną metodą MES. Dzięki wprowadzeniu równań równowagi i relacji konstytutywnych w funkcję kosztu sieci te są w stanie poprawnie przewidywać rozwiązania bez konieczności generowania siatki elementów skończonych. Rozwiązanie takiego zagadnienia zajmuje ok. 10 minut, czyli dłużej niż pojedyncza zadanie MES, lecz czas ten pozostaje akceptowalny i pozwala na iteracyjne dostrajanie oraz optymalizację architektury sieci i jej hiperparametrów przed przystąpieniem do problemu odwrotnego.

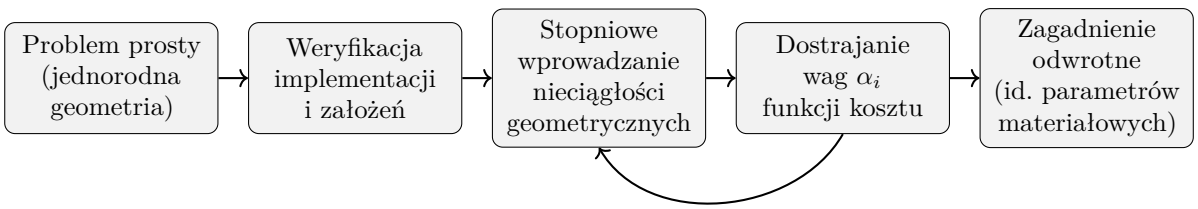
W przypadku problemu odwrotnego celem była identyfikacja parametrów modelu Yeoha na podstawie danych uzyskanych z MES, symulujących te z cyfrowej korelacji obrazu oraz historii obciążenia próbki. Parametry materiałowe zostały włączone do wektora zmiennych optymalizowanych, wraz z wagami sieci neuronowej. Sieć jednocześnie uczy się aproksymować pola przemieszczeń, naprężeń a także wartości parametrów materiałowych. Dzięki temu otrzymujemy pojedynczy problem optymalizacji, w którym fizyka, dane oraz parametry materiałowe są ze sobą sprzężone, co odróżnia je od klasycznego podejścia przy wykorzystaniu MES. W tradycyjnym podejściu wyznaczenie parametrów materiałowych polega na dobieraniu kolejnych ich zestawów i przeprowadzaniu kolejnych pełnych zagadnień brzegowo-początkowych metodą elementów skończonych wykonywanego w pętli optymalizacyjnej. Cykl ten może być wykonywany setki, a nawet tysiące razy, co

generuje znaczny koszt obliczeniowy. Trening w zadaniu odwrotnym nieznacznie wzrósł w porównaniu do zadania prostego i wynosił ok. 20 minut, co wynika z większej złożoności przestrzeni poszukiwań. Czas ten jest akceptowalny z praktycznego punktu widzenia, szczególnie w porównaniu z klasycznymi metodami.

Tak jak w rozdziale 7 zastosowanie dwuetapowej strategii optymalizacji okazało się skuteczne. W pierwszej fazie optymalizator pierwszego rzędu Adam, pozwala opuścić obszary o dużej wartości funkcji straty i pozwala kierować rozwiązanie w kierunku globalnego minimum. Gdy proces ulega spowolnieniu, optymalizator drugiego rzędu L-BFGS, wykorzystujący przybliżoną informację o krzywiźnie funkcji straty, umożliwia precyzyjną zbieżność w otoczeniu minimum znalezionego przez Adama.

Jednym z najtrudniejszych wyzwań podczas rozwiązywania zadań w niniejszym rozdziale okazało się dobranie współczynników wagowych α_i poszczególnych składników funkcji kosztu. Problem ten jest dobrze znany w literaturze i wraz z długim czasem treningu stanowi główną barierę ograniczającą szersze zastosowanie komercyjne. Dobór wag opiera się na intuicji i doświadczeniu zdobytym w pracy nad konkretnym typem zagadnienia brzegowego. Proces ten można częściowo zautomatyzować przy użyciu narzędzi do optymalizacji hiperparametrów takich jak Optuna, ale generalnie nie istnieje żadna uniwersalna metoda doboru tych wag.

Pisząc o intuicji i doświadczeniu, autor poleca zacząć od stopniowego zwiększania złożoności zadania. Chcąc wyznaczyć parametry hipersprężystości w rozciąganej próbce poleca się zacząć od problemu prostego w celu weryfikacji poprawności zaimplementowania wszystkich założeń. Upewniając się, że sieć generuje poprawne rozwiązanie, należy stopniowo wprowadzać i powiększać nieciągłości geometryczne oraz dopasowywać współczynniki wagowe funkcji kosztu. Dopiero po dobrym opanowaniu zadania prostego można przejść do bardziej złożonego zadania odwrotnego i wyznaczyć parametry materiałowe.



Rysunek 8.13. Zalecana strategia stopniowego zwiększania złożoności zadania przy rozwiązywaniu zagadnienia odwrotnego w celu wyznaczenia wartości parametrów materiałowych

Kluczowe do poprawnego rozwiązania zadania wyznaczenia parametrów materiałowych jest połączenie historii obciążenia z polami przemieszczeń. Historia obciążenia stanowi standardowy warunek brzegowy, natomiast pole przemieszczeń dostarcza informacji o lokalnych gradientach deformacji, szczególnie w miejscach występowania niejednorodności geometrycznych.

Zaprezentowana architektura PINN wykazała się odpornością na szum pomiarowy o poziomie 1%, co sugeruje, że podejście może być użyte w rzeczywistych danych eksperymentalnych.

Dalszy rozwój powinien skupić się na automatyzacji doboru funkcji wagowych oraz na rozszerzeniu algorytmu na bardziej złożone modele konstytutywne hipersprężystości. Dla klasycznych modeli hipersprężystości zaproponowane podejście powinno się przenieść bez większych trudności. Bardziej interesującym kierunkiem jest natomiast integracja z konstytutywnymi sieciami neuronowymi (CANN, *Constitutive Artificial Neural Networks*), w których funkcja jednostkowej energii sprężystości mogłaby być optymalizowana równolegle z rozwiązaniem zagadnienia brzegowego.

9. Podsumowanie i wnioski ogólne

Celem rozprawy było opracowanie oraz weryfikacja metod uczenia maszynowego z więzami fizycznymi, służących do modelowania konstytutywnego i rozwiązywania zagadnień brzegowych teorii hipersprężystości. Przeprowadzone badania potwierdziły tezę pracy: wykazano, że klasyczne algorytmy uczenia maszynowego, pozbawione wiedzy kierunkowej, nie są w stanie poprawnie odtworzyć relacji konstytutywnych hipersprężystości, natomiast modele uwzględniające więzy mechaniki ośrodków ciągłych — regresja z odpowiednio dobraną bazą funkcyjną, konstytutywne sieci neuronowe, CANN [14], [15] oraz sieci neuronowe informowane fizyką, PINN [2] — pozwalają zarówno określać relacje konstytutywne, jak i rozwiązywać zagadnienia brzegowe oraz identyfikować parametry materiałowe wyłącznie na podstawie danych przemieszczeniowych i sił reakcji podporowych. Poniżej zestawiono wnioski o charakterze ogólnym oraz oryginalny wkład pracy, wnioski szczegółowe zamieszczono w poszczególnych rozdziałach rozprawy.

Wnioski ogólne

- Spośród kilkunastu przebadanych algorytmów klasycznego uczenia maszynowego żaden nie spełnił wymagań modelowania konstytutywnego bez wprowadzenia wiedzy kierunkowej. Typowe metryki uczenia maszynowego (RMSE, R^2) nie uwzględniają spełnienia więzów fizycznych ani zdolności do ekstrapolacji — wartości R^2 rzędu 0.97–0.999, uznawane za bardzo dobre w klasycznych zastosowaniach, okazują się niewystarczające w modelowaniu hipersprężystości, w którym kluczowa jest ekstrapolacja poza zakres danych treningowych.
- Spośród metod klasycznych do modelowania związków konstytutywnych najlepiej nadają się te generujące ciągłe, interpretowalne funkcje regresji zdolne do ekstrapolacji — regresja liniowa oraz regresor maszyny wektorów nośnych z jądrem liniowym. Metody drzewiaste, niezdolne do interpolacji wartości spoza zbioru treningowego, okazały się w tym zadaniu nieprzydatne.
- Wykazano, że modele zależne wyłącznie od pierwszego niezmiennika deformacji izochorycznej (neo-Hooke'a, Yeoha) są niewystarczające do poprawnego odwzorowania pełnego zestawu stanów deformacji z testów doświadczalnych Treloara [33] — w szczególności test dwuosiowego rozciągania, najczulszy na drugi niezmiennik, wymaga uwzględnienia członów zależnych od $\bar{I}_2$.

- Metody regresji i sieci neuronowe prowadzą do odmiennych, lecz poprawnych modeli materiałowych: regresja z regularyzacją Lasso preferuje człony wielomianowe, podczas gdy sieć neuronowa równomierniej rozkłada wkłady poszczególnych funkcji bazowych.
- W zagadnieniach rozwiązywanych metodą PINN normalizacja zmiennych wejściowych i wyjściowych okazała się kluczowa dla zbieżności. Wrażliwość na skalowanie zależy od wymiaru zadania — w zagadnieniu jednowymiarowym (belka) wystarczyła normalizacja współrzędnej, podczas gdy w zagadnieniu dwuwymiarowym (tarcza) niezbędne było skalowanie zarówno współrzędnych, jak i wyjść sieci. Sprowadzenie wielkości do porównywalnych zakresów ogranicza u źródła problem niezbalansowanych składników funkcji straty.
- Dla zagadnień brzegowych liniowej teorii sprężystości metoda PINN, mimo poprawnych wyników, nie oferuje przewagi nad metodą elementów skończonych pod względem dokładności ani czasu obliczeń, a jej potencjał ujawnia się dopiero w zagadnieniach odwrotnych.
- Dokładność rozwiązania PINN zależy od liczby punktów residuum w sposób analogiczny do zależności dokładności MES od gęstości dyskretyzacji — w obu podejściach równania wymusza się w skończonej liczbie punktów, a zbyt rzadkie próbkowanie domeny uniemożliwia uzyskanie zbieżności.
- Dwuetapowa strategia optymalizacji (Adam [44] + L-BFGS [45]) okazała się skuteczna we wszystkich rozpatrywanych zadaniach. Optymalizator pierwszego rzędu Adam pozwala opuścić obszary o dużej wartości funkcji straty, a optymalizator drugiego rzędu L-BFGS, wykorzystujący przybliżoną informację o krzywiźnie, umożliwia precyzyjną zbieżność w otoczeniu znalezionego minimum. Zaobserwowano przy tym wrażliwość wyniku na punkt startowy fazy L-BFGS — częściowy trening Adamem może prowadzić sieć do obszaru przestrzeni parametrów, z którego trudniej osiągnąć minimum globalne.
- W zagadnieniu odwrotnym parametry materiałowe włączono do wektora zmiennych optymalizowanych łącznie z wagami sieci, dzięki czemu fizyka, dane pomiarowe i parametry konstytutywne są wyznaczane w ramach pojedynczego, sprzężonego problemu optymalizacji. Odróżnia to proponowane podejście od klasycznej analizy odwrotnej opartej na MES [9], w której pełne zagadnienie brzegowe rozwiązuje się wielokrotnie w pętli optymalizacyjnej, generując znaczny koszt obliczeniowy.

- Głównym wyzwaniem praktycznym pozostaje dobór współczynników wagowych poszczególnych składników funkcji kosztu sieci PINN. Problem ten, obok czasu treningu, stanowi istotną barierę szerszego, komercyjnego zastosowania metody i opiera się obecnie głównie na intuicji oraz doświadczeniu w pracy nad danym typem zagadnienia.

Oryginalny wkład rozprawy

- Przeprowadzono usystematyzowane studium dostępnych metod klasycznego uczenia maszynowego pod kątem ich przydatności do modelowania konstytutywnego hipersprężystości, wskazując metody właściwe oraz wykazując ograniczenia pozostałych.
- Opracowano moduł regresji liniowej umożliwiający analizowanie pełnych, różnorodnych stanów deformacji oraz wprowadzono pojęcie macierzy naprężeniowej cech, której szczegółowe działanie przedstawiono w studium przypadku. W module tym połączono modele zgodne z propozycją Hartmanna i Neffa [120], wprowadzając właściwą potęgę niezmiennika $\bar{I}_2^{3/2}$. W obu modelach — regresji liniowej i sieci neuronowej — wejścia znormalizowano, co wymagało zastosowania odwrotności normalizacji do współczynników materiałowych.
- Wykazano, że metody regresji i sieci neuronowej prowadzą do niezależnych, lecz poprawnych modeli materiałowych, a odzyskanie interpretowalnych współczynników materiałowych jest możliwe również w modelu opartym na sieci neuronowej z normalizacją wejść.
- Wykazano, że sieć PINN dla zagadnień teorii sprężystości można równoważnie zaimplementować w czystym środowisku PyTorch oraz z użyciem biblioteki DeepXDE, uzyskując tożsame wyniki. Przeprowadzono szczegółową analizę zbieżności, obejmującą wpływ normalizacji zmiennych, architektury sieci (FNN, PFNN), sposobu narzucania warunków brzegowych (miękkie, twarde) oraz strategii optymalizacji na dokładność i koszt obliczeniowy rozwiązania.
- Zaobserwowano, że dla rozpatrywanego zagadnienia tarczowego sam optymalizator L-BFGS pozwala uzyskać dokładność porównywalną z pełnym treningiem dwuetapowym w czasie krótszym o ponad dwa rzędy wielkości, co sugeruje wykonalność tego typu obliczeń również na konsumenckich kartach graficznych.

- Zaadaptowano przedstawione w pracy Songa i Jina [145] podejście do identyfikacji parametrów hipersprężystości metodą PINN — w którym parametry materiałowe wyznacza się łącznie z wagami sieci na podstawie pól przemieszczeń oraz historii obciążenia — i zastosowano je do identyfikacji parametrów modelu Yeoha w zagadnieniu rozciąganej tarczy z otworem. Głównym wkładem własnym jest opracowanie samodzielnego, modułowego i rozszerzalnego kodu realizującego to podejście, w miejsce jednorazowej, dedykowanej implementacji, co umożliwia jego stosowanie do innych modeli konstytutywnych i zagadnień brzegowych. Dostosowano je przy tym do modelu Yeoha oraz rozpatrywanej tarczy z otworem.
- Zaproponowano strategię stopniowego zwiększania złożoności zadania — od zagadnienia prostego, poprzez wprowadzanie nieciągłości geometrycznych i dostrajanie wag funkcji kosztu, po zagadnienie odwrotne — jako praktyczny sposób postępowania przy identyfikacji parametrów hipersprężystości metodą PINN.

Kierunki dalszych badań

- Rozwinięcie metryk oraz funkcji kosztu uwzględniających więzy fizyczne i zdolność do ekstrapolacji, np. poprzez trening Sobolewa kontrolujący zgodność nachylenia krzywych naprężenie–wydłużenie na krańcach przedziału lub w całym obszarze [123], [124].
- Automatyzację doboru współczynników wagowych funkcji kosztu w sieciach PINN, np. z wykorzystaniem narzędzi optymalizacji hiperparametrów lub adaptacyjnych schematów równoważenia składników straty [34], [136].
- Rozszerzenie podejścia PINN na bardziej złożone modele konstytutywne hipersprężystości — dla modeli klasycznych przeniesienie metody powinno nastąpić bez większych trudności.
- Integrację sieci PINN z konstytutywnymi sieciami neuronowymi (CANN), w których funkcja jednostkowej energii sprężystości byłaby optymalizowana równolegle z rozwiązaniem zagadnienia brzegowego.
- Weryfikację proponowanych metod na rzeczywistych danych eksperymentalnych, m.in. podstawowych testów wytrzymałościowych oraz pochodzących z cyfrowej korelacji obrazu.

Bibliografia

[1] I. Goodfellow, Y. Bengio i A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016.

[2] M. Raissi, P. Perdikaris i G. E. Karniadakis, „Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations", *Journal of Computational Physics*, t. 378, s. 686–707, 2019.

[3] C. Rackauckas, Y. Ma, J. Martensen i in., „Universal Differential Equations for Scientific Machine Learning", 2020.

[4] K. Hornik, M. Stinchcombe i H. White, „Multilayer feedforward networks are universal approximators", *Neural Networks*, t. 2, nr. 5, s. 359–366, 1989.

[5] J. H. Garrett, „Where and Why Artificial Neural Networks Are Applicable in Civil Engineering", *Journal of Computing in Civil Engineering*, t. 8, nr. 2, s. 129–130, 1994.

[6] I. Flood i N. Kartam, „Neural Networks in Civil Engineering. I: Principles and Understanding", *Journal of Computing in Civil Engineering*, t. 8, nr. 2, s. 131–148, 1994.

[7] H. Adeli, „Neural Networks in Civil Engineering: 1989–2000", *Computer-Aided Civil and Infrastructure Engineering*, t. 16, nr. 2, s. 126–142, 2001.

[8] J. Ghaboussi, J. H. Garrett i X. Wu, „Knowledge-Based Modeling of Material Behavior with Neural Networks", *Journal of Engineering Mechanics*, t. 117, nr. 1, s. 132–153, 1991.

[9] O. C. Zienkiewicz i R. L. Taylor, *The Finite Element Method for Solid and Structural Mechanics*, Seventh, O. C. Zienkiewicz, R. L. Taylor i D. Fox, red. Oxford: Butterworth-Heinemann, 2014.

[10] M. Lefik i B. A. Schrefler, „Artificial neural network as an incremental non-linear constitutive model for a finite element code", *Computer Methods in Applied Mechanics and Engineering*, t. 192, nr. 28-30, s. 3265–3283, 2003.

[11] Y. M. Hashash, S. Jung i J. Ghaboussi, „Numerical implementation of a neural network based material model in finite element analysis", *International Journal for Numerical Methods in Engineering*, t. 59, nr. 7, s. 989–1005, 2004.

[12] R. Walentyński, *Wybrane zagadnienia mechaniki konstrukcji w ujęciu algebry komputerowej i inteligencji obliczeniowej*. Gliwice: Wydawnictwo Politechniki Śląskiej, 2024.

[13] E. Haghighat, M. Raissi, A. Moure, H. Gomez i R. Juanes, „A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics", *Computer Methods in Applied Mechanics and Engineering*, t. 379, 2021.

[14] K. Linka, M. Hillgärtner, K. P. Abdolazizi, R. C. Aydin, M. Itskov i C. J. Cyron, „Constitutive artificial neural networks: A fast and general approach to predictive data-driven constitutive modeling by deep learning", *Journal of Computational Physics*, t. 429, 2021.

[15] K. Linka i E. Kuhl, „A new family of Constitutive Artificial Neural Networks towards automated model discovery", *Computer Methods in Applied Mechanics and Engineering*, t. 403, 2023.

[16] O. H. Yeoh, „Some Forms of the Strain Energy Function for Rubber", *Rubber Chemistry and Technology*, t. 66, nr. 5, s. 754–771, 1993.

[17] *Kaggle: Your Home for Data Science*. adr.: `https://www.kaggle.com/`.

[18] *Dataset Search*. adr.: `https://datasetsearch.research.google.com/`.

[19] *Datasets - UCI Machine Learning Repository*. adr.: `https://archive.ics.uci.edu/datasets`.

[20] *DataHub - a complete solution for Open Data Platforms, Data Catalogs, Data Lakes and Data Management*. adr.: `https://datahub.io/collections`.

[21] A. Cena, M. Gągolewski i M. Bartoszuk, *Przetwarzanie i analiza danych w języku Python*. Warszawa: Wydawnictwo Naukowe PWN, 2016.

[22] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA: MIT Press, 2012.

[23] G. Schwarz, „Estimating the Dimension of a Model", *Annals of Statistics*, t. 6, nr. 2, s. 461–464, 1978.

[24] H. Akaike, „A New Look at the Statistical Model Identification", *IEEE Transactions on Automatic Control*, t. 19, nr. 6, s. 716–723, 1974.

[25] *scikit-learn: machine learning in Python — scikit-learn 1.7.0 documentation*. adr.: `https://scikit-learn.org/stable/`.

[26] J. Sudyka i G. Mazurek, „Wyznaczanie temperatury warstw asfaltowych w pomiarach FWD i TSD z wykorzystaniem uczenia maszynowego”, *Roads and Bridges – Drogi i Mosty*, t. 24, nr. 3, s. 267–282, 2025.

[27] C. Cortes, V. Vapnik i L. Saitta, „Support-vector networks”, *Machine Learning*, t. 20, nr. 3, s. 273–297, 1995.

[28] H. Drucker, „Support Vector Regression Machines”, *Advances in Neural Information Processing Systems*, 1996.

[29] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

[30] C. C. Aggarwal, A. Hinneburg i D. A. Keim, „On the Surprising Behavior of Distance Metrics in High Dimensional Space”, J. Van den Bussche i V. Vianu, red., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, s. 420–434.

[31] F. Galton, „Regression Towards Mediocrity in Hereditary Stature”, *The Journal of the Anthropological Institute of Great Britain and Ireland*, t. 15, s. 246–263, 1886.

[32] K. Pearson, „Mathematical Contributions to the Theory of Evolution. III. Regression, Heredity, and Panmixia”, *Philosophical Transactions of the Royal Society of London. Series A*, t. 187, s. 253–318, 1895.

[33] L. Treloar, „Stress-strain data for vulcanised rubber under various types of deformation”, *Transactions of the Faraday Society*, t. 40, s. 59, 1944.

[34] R. Bischof i M. A. Kraus, „Multi-Objective Loss Balancing for Physics-Informed Deep Learning”, *Computer Methods in Applied Mechanics and Engineering*, t. 439, s. 117 914, 2025.

[35] J. Tabor, M. Śmieja, Ł. Struski, P. Spurek i M. Wołczyk, *Głębokie uczenie: wprowadzenie*, pol. Wydawnictwo Naukowe Helion, 2022.

[36] F. Rosenblatt, „The perceptron: A probabilistic model for information storage and organization in the brain”, *Psychological Review*, t. 65, nr. 6, s. 386–408, 1958.

[37] V. Nair i G. E. Hinton, „Rectified linear units improve restricted Boltzmann machines”, w *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010, s. 807–814.

[38] X. Glorot i Y. Bengio, „Understanding the difficulty of training deep feedforward neural networks”, w *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Y. W. Teh i M. Titterington, red., ser. Proce-

edings of Machine Learning Research, t. 9, Chia Laguna Resort, Sardinia, Italy: PMLR, 2010, s. 249–256.

[39] K. He, X. Zhang, S. Ren i J. Sun, „Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", w *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015, s. 1026–1034.

[40] G. Cybenko, „Approximation by superpositions of a sigmoidal function", *Mathematics of Control, Signals, and Systems*, t. 2, nr. 4, s. 303–314, 1989.

[41] *Linear elastostatic equation over a 2D square domain — DeepXDE 0.1.dev1+gb8d69c431 documentation*. adr.: `https://deepxde.readthedocs.io/en/latest/demos/pinn_forward/elasticity.plate.html`.

[42] A. Paszke, S. Gross, S. Chintala i in., „Automatic differentiation in PyTorch", w *NIPS 2017 Workshop on Autodiff*, 2017.

[43] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York: Springer, 2006.

[44] D. P. Kingma i J. L. Ba, „Adam: A Method for Stochastic Optimization", w *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, International Conference on Learning Representations, ICLR, 2015.

[45] D. C. Liu i J. Nocedal, „On the limited memory BFGS method for large scale optimization", *Mathematical Programming*, t. 45, nr. 1-3, s. 503–528, 1989.

[46] H. Robbins i S. Monro, „A Stochastic Approximation Method", *Annals of Mathematical Statistics*, t. 22, nr. 3, s. 400–407, 1951.

[47] T. Hastie, R. Tibshirani i J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2 wyd. New York: Springer, 2009.

[48] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever i R. Salakhutdinov, „Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research*, t. 15, nr. 56, s. 1929–1958, 2014.

[49] R. M. Bowen, *Introduction to continuum mechanics for engineers (Mathematical concepts and methods in science and engineering)*. Springer, 1989, t. 39.

[50] M. E. Gurtin, *An Introduction to Continuum Mechanics*. Academic Press, 1982, t. 158.

[51] P. Haupt, *Continuum Mechanics and Theory of Materials* (Advanced Texts in Physics). Berlin, Heidelberg: Springer Berlin Heidelberg, 2002.

[52] A. Krawietz, *Materialtheorie*. Springer Berlin Heidelberg, 1986.

[53] J. Ostrowska-Maciejewska, *Mechanika ciał odkształcalnych* (Biblioteka Mechaniki Stosowanej: Podręczniki i Zastosowania Inżynierskie). Wydawnictwo Naukowe PWN, 1994.

[54] M. Silhavy, *The Mechanics and Thermodynamics of Continuous Media*. 2002.

[55] D. R. Smith, *An Introduction to Continuum Mechanics — after Truesdell and Noll* (Solid Mechanics and Its Applications). Dordrecht: Springer Netherlands, 1993, t. 22.

[56] C. Truesdell i R. Toupin, „The Classical Field Theories", *HDP*, t. 2, s. 226–858, 1960.

[57] C. Truesdell i W. Noll, „The Non-Linear Field Theories of Mechanics", *The Non-Linear Field Theories of Mechanics*, 2004.

[58] C. Wang i C. Truesdell, *Introduction to Rational Elasticity*. Leyden: Noordhoff, 1973.

[59] S. S. Antman, *Nonlinear Problems of Elasticity*. Springer, 2005, t. 107, s. 750.

[60] J. Bonet i R. D. Wood, *Nonlinear Continuum Mechanics for Finite Element Analysis*. Cambridge: Cambridge University Press, 2008, s. 1–318.

[61] P. G. Ciarlet, *Mathematical elasticity*. Amsterdam-Tokyo: North-Holland, 1988.

[62] A. E. Green i W. Zerna, *Non-linear elastic deformations*. Chichester: Horwood, 1968.

[63] A. E. Green i J. E. Adkins, „Large Elastic Deformations", *Physics Today*, t. 24, nr. 12, s. 57, 1971.

[64] G. A. Holzapfel, *Nonlinear solid mechanics : a continuum approach for engineering*. John Wiley & Sons, 2006, s. 455.

[65] J. E. Marsden i T. J. R. Hughes, *Mathematical Foundations of Elasticity*. New York: Prentice-Hall, Englewood Cliffs, 1983, s. 946.

[66] R. W. Ogden, *Non-linear elastic deformations*. Ellis Horwood, Chichester, 1984.

[67] S. Jemioło, *Studium hipersprężystych własności materiałów izotropowych - modelowanie i implementacja numeryczna*. Warsaw: Warsaw University of Technology Publishing House, 2002.

[68] S. Jemioło, *Relacje konstytutywne hipersprężystości*, Polska Akademia Nauk, red. Warszawa: Komitet Inżynierii Lądowej i Wodnej PAN, 2016.

[69] S. Jemioło i M. Gajewski, *Hipersprężystoplastyczność* (Monografie Zakładu Wytrzymałości Materiałów Teorii Sprężystości i Plastyczności t. 4). Warszawa: Zakład Wytrzymałości Materiałów, Teorii Sprężystości i Plastyczności, Wydział Inżynierii Lądowej Politechniki Warszawskiej, 2014.

[70] S. Jemioło i M. Gajewski, *Sprężystość i hipersprężystość : modelowanie i zastosowania. Cztery typy płaskiej anizotropii na przykładzie modelu kompozytu włóknistego.* (Monografie Zakładu Wytrzymałości Materiałów Teorii Sprężystości i Plastyczności t. 1), Wyd. 2, uz. Warszawa: Zakład Wytrzymałości Materiałów, Teorii Sprężystości i Plastyczności, Wydział Inżynierii Lądowej Politechniki Warszawskiej, 2016, s. 45–56.

[71] S. Jemioło i C. Suchocki, *Hipersprężystość i jej modyfikacje. Zarys teorii, pseudo-hipersprężystość i quasi liniowa lepko-sprężystość*, 1 wyd. Warszawa: Oficyna Wydawnicza Politechniki Warszawskiej, 2018, s. 196.

[72] S. Jemioło i A. Ł. Franus, *Izotropowe modele konstytutywne hipersprężystości. Wyznaczanie parametrów materiałowych i implementacja numeryczna.* Warszawa: Zakład Wytrzymałości Materiałów, Teorii Sprężystości i Plastyczności, Wydział Inżynierii Lądowej Politechniki Warszawskiej, 2019.

[73] M. Miecznikowski, „Modelowanie łożysk elastomerowych w zakresie dużych deformacji", Praca dyplomowa magisterska, Politechnika Warszawska, Wydział Inżynierii Lądowej, Warszawa, 2020.

[74] S. Jemioło, „Modele konstytutywne kompozytowych materiałów z włóknami i modele siatek do zbrojenia nawierzchni", *Instytut Mechaniki Konstrukcji Inżynierskich. Politechnika Warszawska.*, s. 5–38, 2005.

[75] L. Yang, D. Zhang i G. E. Karniadakis, „Physics-Informed Generative Adversarial Networks for Stochastic Differential Equations", *SIAM Journal on Scientific Computing*, t. 42, nr. 1, A292–A317, 2020.

[76] M. Gajewski, *Badania lepiszczy asfaltowych w reometrze dynamicznego ścinania - relacje konstytutywne lepko-sprężystości, hipersprężystości i lepko-hipersprężystości.* Warszawa: Oficyna Wydawnicza Politechniki Warszawskiej, 2018, s. 236.

[77] R. S. Rivlin, „Large elastic deformations of isotropic materials. II. Some uniqueness theorems for pure, homogeneous deformation", *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, t. 240, nr. 822, s. 491–508, 1948.

[78] P. J. Flory, „Thermodynamic relations for high elastic materials", *Transactions of the Faraday Society*, t. 57, nr. 0, s. 829–838, 1961.

[79] S. Jemioło, R. Szczerba i M. Gajewski, „Zastosowanie hipersprężystości i MES w modelowaniu mostowych łożysk elastomerowych", w *Współczesna mechanika konstrukcji w projektowaniu inżynierskim*, t. 92, Warszawa: Komitet Inżynierii Lądowej i Wodnej PAN, 2015, s. 97–122.

[80] Hibbit, Karlsson i Sorensen, *ABAQUS Theory manual*, Pawtucket, 2000.

[81] ANSYS Inc., *ANSYS Mechanical APDL Theory Reference*, Canonsburg, PA, USA, 2023.

[82] MSC Software Corporation, *Marc Volume A: Theory and User Information*, Newport Beach, CA, USA, 2023.

[83] J. C. Simo, R. L. Taylor i K. S. Pister, „Variational and projection methods for the volume constraint in finite deformation elasto-plasticity", *Computer Methods in Applied Mechanics and Engineering*, t. 51, nr. 1-3, s. 177–208, 1985.

[84] H. Schreier, J. J. Orteu i M. A. Sutton, *Image correlation for shape, motion and deformation measurements: Basic concepts, theory and applications*. Springer New York, NY, 2009, s. 1–321.

[85] S. N. Olufsen, M. E. Andersen i E. Fagerholt, „μDIC: An open-source toolkit for digital image correlation", *SoftwareX*, t. 11, 2020.

[86] V. Belloni, R. Ravanelli, A. Nascetti, M. D. Rita, D. Mattei i M. Crespi, „py2DIC: A New Free and Open Source Software for Displacement and Strain Measurements in the Field of Experimental Mechanics", *Sensors*, t. 19, nr. 18, 2019.

[87] J. Wojciechowski, M. Gajewski i C. Ajdukiewicz, „Procedura wyznaczania parametrów sprężysto-plastyczności ze wzmocnieniem izotropowym na podstawie badań doświadczalnych przy zastosowaniu systemu optyczno-pomiarowego ARAMIS", w *Deformacje i wytrzymałość materiałów i elementów konstrukcji*, Warszawa, 2013, rozd. 6, s. 83–96.

[88] M. Ahmadi, A. McBride, P. Steinmann i P. Saxena, „Plane stress finite element modelling of arbitrary compressible hyperelastic materials", *Acta Mechanica*, t. 236, nr. 7, s. 3975–3994, 2025.

[89] R. T. Rockafellar, *Convex analysis* (Princeton Mathematical Series). Princeton, N. J.: Princeton University Press, 1970.

[90] D. C. Drucker, „A Definition of Stable Inelastic Material", *Applied Mechanics*, t. 26, s. 101–106, 1957.

[91] L. Linden, D. K. Klein, K. A. Kalina, J. Brummund, O. Weeger i M. Kästner, „Neural networks meet hyperelasticity: A guide to enforcing physics", *Journal of the Mechanics and Physics of Solids*, t. 179, 2023.

[92] J. M. Ball, „Constitutive inequalities and existence theorems in nonlinear elastostatics", w *Nonlinear analysis and mechanics: Heriot-Watt symposium*, t. 1, Pitman London, 1977, s. 187–241.

[93] C. B. Morrey, „Quasi-convexity and the lower semicontinuity of multiple integrals", *Pacific Journal of Mathematics*, t. 2, nr. 1, s. 25–53, 1952.

[94] V. Ebbing, *Design of polyconvex energy functions for all anisotropy classes*. Universität Duisburg-Essen, 2010.

[95] J. M. Ball, *Discontinuous equilibrium solutions and cavitation in nonlinear elasticity*, London, 1982.

[96] B. Amos, L. Xu i J. Zico Kolter, „Input Convex Neural Networks", w *Proceedings of the 34th International Conference on Machine Learning*, 2017, s. 146–155.

[97] D. K. Klein, M. Fernández, R. J. Martin, P. Neff i O. Weeger, „Polyconvex anisotropic hyperelasticity with neural networks", *Journal of the Mechanics and Physics of Solids*, t. 159, 2022.

[98] J. M. Ball, „Convexity conditions and existence theorems in nonlinear elasticity", *Archive for Rational Mechanics and Analysis*, t. 63, nr. 4, s. 337–403, 1976.

[99] A. Mielke, „Necessary and sufficient conditions for polyconvexity of isotropic functions", *Journal of Convex Analysis*, t. 12, nr. 2, s. 291–314, 2005.

[100] J. G. Sambou i E. Diouf, „An equivalence between the rank 1 convexity and polyconvexity of energy functions on SL+(3)", *International Journal of Mathematics Trends and Technology*, t. 66, nr. 9, s. 17–25, 2020.

[101] S. J. Spector, „A Note on the Convexity of $\mathbf{C} \to h(\det \mathbf{C})$", *Journal of Elasticity*, t. 118, nr. 2, s. 251–256, 2015.

[102] G.-L. Geuken, P. Kurzeja, D. Wiedemann i J. Mosler, „Input convex neural networks: universal approximation theorem and implementation for isotropic polyconvex hyperelastic energies", *Journal of the Mechanics and Physics of Solids*, t. 203, 2025.

[103] H. S. Suh, C. Kweon, B. Lester, S. Kramer i W. C. Sun, „A publicly available PyTorch-ABAQUS UMAT deep-learning framework for level-set plasticity", *Mechanics of Materials*, t. 184, 2023.

[104] J. W. Backus, R. J. Beeber, S. Best i in., „The FORTRAN automatic coding system", *Proceedings of the Western Joint Computer Conference, IRE-AIEE-ACM 1957*, s. 188–198, 1957.

[105] D. E. Rumelhart, G. E. Hinton i R. J. Williams, „Learning representations by back-propagating errors", *Nature*, t. 323, nr. 6088, s. 533–536, 1986.

[106] H. Moon, D. Park, H. Cho, H. K. Noh, J. H. Lim i S. Ryu, „Physics-informed neural network-based discovery of hyperelastic constitutive models from extremely scarce data", *Computer Methods in Applied Mechanics and Engineering*, t. 446, s. 118 258, 2025.

[107] I. Wolfram Research, *Mathematica, Version 14.3*. adr.: `https://www.wolfram.com/mathematica`.

[108] M. Flaschel, S. Kumar i L. De Lorenzis, „Automated discovery of generalized standard material models with EUCLID", *Computer Methods in Applied Mechanics and Engineering*, t. 405, 2023.

[109] J.-H. Urrea–Quintero, D. Anton, L. De Lorenzis i H. Wessels, „Automated Constitutive Model Discovery by Pairing Sparse Regression Algorithms with Model Selection Criteria", *Computer Methods in Applied Mechanics and Engineering*, t. 449, 2025.

[110] K. Linka, S. R. St. Pierre i E. Kuhl, „Automated model discovery for human brain using Constitutive Artificial Neural Networks", *Acta Biomaterialia*, t. 160, s. 134–151, 2023.

[111] P. Thakolkaran, A. Joshi, Y. Zheng, M. Flaschel, L. De Lorenzis i S. Kumar, „NN-EUCLID: Deep-learning hyperelasticity without stress data", *Journal of the Mechanics and Physics of Solids*, t. 169, 2022.

[112] G. Montúfar, R. Pascanu, K. Cho i Y. Bengio, „On the Number of Linear Regions of Deep Neural Networks", *Advances in Neural Information Processing Systems*, t. 2, s. 2924–2932, 2014.

[113] H. Demiray, „A note on the elasticity of soft biological tissues", *Journal of Biomechanics*, t. 5, nr. 3, s. 309–311, 1972.

[114] M. Abadi, P. Barham, J. Chen i in., „TensorFlow: A system for large-scale machine learning", *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016*, s. 265–283, 2016.

[115] *Treolar Dataset*. adr.: `https://www.kaggle.com/datasets/swordfisho/treolar-dataset`.

[116] P. Vaníček, E. W. Grafarend i M. Berber, „Short note: Strain invariants", *Journal of Geodesy*, t. 82, nr. 4-5, s. 263–268, 2008.

[117] T. Sussman i K. J. Bathe, „A finite element formulation for nonlinear incompressible elastic and inelastic analysis", *Computers & Structures*, t. 26, nr. 1-2, s. 357–409, 1987.

[118] M. Mooney, „A Theory of Large Elastic Deformation", *Journal of Applied Physics*, t. 11, nr. 9, s. 582–592, 1940.

[119] V. L. Biderman, *Calculation of Rubber Parts*. Moscow: Rascheti na Prochnost, 1958.

[120] S. Hartmann i P. Neff, „Polyconvexity of generalized polynomial-type hyperelastic strain energy functions for near-incompressibility", *International Journal of Solids and Structures*, t. 40, nr. 11, s. 2767–2791, 2003.

[121] *SymPy*. adr.: `https://www.sympy.org/en/index.html`.

[122] S. K. Melly, L. Liu, Y. Liu i J. Leng, „Modified Yeoh model with improved equibiaxial loading predictions", *Acta Mechanica*, t. 233, nr. 2, s. 437–453, 2022.

[123] W. M. Czarnecki, S. Osindero, M. Jaderberg, G. Swirszcz i R. Pascanu, „Sobolev Training for Neural Networks", *Advances in Neural Information Processing Systems*, s. 4279–4288, 2017.

[124] N. N. Vlassis i W. C. Sun, „Sobolev training of thermodynamic-informed neural networks for interpretable elasto-plasticity models with level set hardening", *Computer Methods in Applied Mechanics and Engineering*, t. 377, 2021.

[125] M. Raissi, A. Yazdani i G. E. Karniadakis, „Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations", *Science*, t. 367, nr. 6481, s. 1026–1030, 2020.

[126] A. M. Tartakovsky, C. O. Marrero, P. Perdikaris, G. D. Tartakovsky i D. Barajas-Solano, „Physics-Informed Deep Neural Networks for Learning Parameters and Constitutive Relationships in Subsurface Flow Problems", *Water Resources Research*, t. 56, nr. 5, e2019WR026731, 2020.

[127] Q. Z. He i A. M. Tartakovsky, „Physics-Informed Neural Network Method for Forward and Backward Advection-Dispersion Equations", *Water Resources Research*, t. 57, nr. 7, 2021.

[128] Y. Chen, L. Lu, G. E. Karniadakis i L. D. Negro, „Physics-informed neural networks for inverse problems in nano-optics and metamaterials", *Optics Express*, t. 28, nr. 8, s. 11 618, 2020.

[129] D. Zhang, L. Lu, L. Guo i G. E. Karniadakis, „Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems", *Journal of Computational Physics*, t. 397, 2018.

[130] D. Zhang, L. Guo i G. E. Karniadakis, „Learning in Modal Space: Solving Time-Dependent Stochastic PDEs Using Physics-Informed Neural Networks", *SIAM Journal on Scientific Computing*, t. 42, nr. 2, A639–A665, 2019.

[131] L. Lu, X. Meng, Z. Mao i G. E. Karniadakis, „DeepXDE: A deep learning library for solving differential equations", *SIAM Review*, t. 63, s. 208–228, 2020.

[132] *Poisson equation in 1D with Dirichlet boundary conditions — DeepXDE 0.1.dev1+gf05d9f051 documentation*. adr.: `https://deepxde.readthedocs.io/en/latest/demos/pinn_forward/poisson.1d.dirichlet.html`.

[133] W. Nowacki, *Teoria sprężystości*, 1973.

[134] S. Jemioło, M. Gajewski, Kamińska Inez i in., *Termosprężystość i przepływ ciepła w materiałach anizotropowych*. Zakład Wytrzymałości Materiałów, Teorii Sprężystości i Plastyczności. Wydział Inżynierii Lądowej PW, 2015.

[135] A. Szwed i S. Jemioło, „Teoria Sprężystości i Plastyczności – rękopis do Wykładu".

[136] S. Wang, Y. Teng i P. Perdikaris, „Understanding and mitigating gradient flow pathologies in physics-informed neural networks", *SIAM Journal on Scientific Computing*, t. 43, nr. 5, s. 3055–3081, 2021.

[137] *Netron*. adr.: `https://netron.app/`.

[138] C. Wu, M. Zhu, Q. Tan, Y. Kartha i L. Lu, „A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks", *Computer Methods in Applied Mechanics and Engineering*, t. 403, 2023.

[139] A. Tomaszewska i D. Reznikov, „Optimization of constitutive law for objective numerical modeling of knitted fabric", *Journal of the Mechanics and Physics of Solids*, t. 196, 2025.

[140] M. D. Gajewski i M. Miecznikowski, „Assessment of the Suitability of Elastomeric Bearings Modeling Using the Hyperelasticity and the Finite Element Method", *Materials*, t. 14, nr. 24, 2021.

[141] T. C. Chu, W. F. Ranson i M. A. Sutton, „Applications of digital-image-correlation techniques to experimental mechanics", *Experimental Mechanics*, t. 25, nr. 3, s. 232–244, 1985.

[142] C. Ajdukiewicz, M. Gajewski i P. Mossakowski, „Zastosowanie systemu optycznej korelacji obrazu Aramis do identyfikacji rys w elementach betonowych", w *TRANSCOMP – XIV International Conference Computer Systems Aided Science, Industry and Transport*, 2010, s. 27–34.

[143] B. Chen, B. Starman, M. Halilovič, L. A. Berglund i S. Coppieters, „Finite Element Model Updating for Material Model Calibration: A Review and Guide to Practice", *Archives of Computational Methods in Engineering*, t. 32, nr. 4, s. 2035–2112, 2024.

[144] Ł. Kowalewski i M. Gajewski, „Assessment of Optimization Methods Used to Determine Plasticity Parameters Based on DIC and back Calculation Methods", *Experimental Techniques*, t. 43, nr. 4, 2019.

[145] S. Song i H. Jin, „Identifying Constitutive Parameters for Complex Hyperelastic Materials using Physics-Informed Neural Networks", *Soft Matter*, t. 20, s. 5915–5926, 2024.

[146] T. Belytschko, J. S. J. Ong, Wing Kam Liu i J. M. Kennedy, „Hourglass control in linear and nonlinear problems", *Computer Methods in Applied Mechanics and Engineering*, t. 43, nr. 3, s. 251–276, 1984.

[147] T. Akiba, S. Sano, T. Yanase, T. Ohta i M. Koyama, „Optuna: A Next-generation Hyperparameter Optimization Framework", *Proceedings of the ACM SIGKDD*

International Conference on Knowledge Discovery and Data Mining, s. 2623–2631, 2019.

[148] M. D. Gajewski, P. Xia, B. Gajewska, J. Pais i M. Miecznikowski, „Estimation of the XGBoost Regression Model Used in the Prediction of Pavement's Mechanical and Geometrical Parameters Based on Static Interpretation of the FWD Test", *Applied Sciences*, t. 15, nr. 24, 2025.

Spis rysunków

Spis tabel

Spis załączników

Załącznik 1. Relacja konstytutywna izotropowych materiałów hipersprężystych w rozkładzie objętościowo-izochorycznym

Niniejszy załącznik przedstawia szczegółowe przekształcenia algebraiczne prowadzące od ogólnych relacji konstytutywnych hipersprężystego materiału izotropowego do postaci wynikających z multiplikatywnego rozdzielenia tensora deformacji $\mathbf{F}$ na część objętościową i izochoryczną. Wyprowadzenie przeprowadza się w opisie Eulera dla tensora naprężenia Cauchy'ego $\boldsymbol{\sigma}$ oraz w opisie Lagrange'a dla II tensora naprężenia Pioli-Kirchhoffa $\mathbf{T}$.

Z1.1. Opis Eulera – tensor naprężenia Cauchy'ego

Punkt wyjścia – eliminacja $\mathbf{B}^2$ twierdzeniem Cayley'a-Hamiltona

Relację konstytutywną w opisie Eulera zapisuje się w postaci:

$$\boldsymbol{\sigma} = \frac{2}{J}\Big(\beta_1\,\mathbf{I} + \beta_2\,\mathbf{B} + \beta_3\,\mathbf{B}^2\Big)\,, \tag{Z1.1}$$

gdzie współczynniki wyrażają się przez pochodne funkcji jednostkowej energii sprężystości $W = W(I_1, I_2, I_3)$:

$$\beta_1 = \alpha_3\,I_3, \quad \beta_2 = \alpha_1 + \alpha_2\,I_1, \quad \beta_3 = -\alpha_2, \quad J = \sqrt{I_3}, \tag{Z1.2}$$

przy czym $\alpha_k = \partial W/\partial I_k$, $k = 1, 2, 3$.

Twierdzenie Cayley'a-Hamiltona dla tensora $\mathbf{B}$ pozwala wyeliminować $\mathbf{B}^2$, ponieważ:

$$\mathbf{B}^3 - I_1\,\mathbf{B}^2 + I_2\,\mathbf{B} - I_3\,\mathbf{I} = \mathbf{0}. \tag{Z1.3}$$

Mnożąc obustronnie przez $\mathbf{B}^{-1}$:

$$\mathbf{B}^2 = I_1\,\mathbf{B} - I_2\,\mathbf{I} + I_3\,\mathbf{B}^{-1}. \tag{Z1.4}$$

Podstawiając (Z1.4) do (Z1.1) i korzystając z (Z1.2):

$$\begin{aligned}\boldsymbol{\sigma} &= \frac{2}{J}\Big(\beta_1\,\mathbf{I} + \beta_2\,\mathbf{B} + \beta_3\Big(I_1\,\mathbf{B} - I_2\,\mathbf{I} + I_3\,\mathbf{B}^{-1}\Big)\Big)\\ &= \frac{2}{J}\Big((\beta_1 - \beta_3\,I_2)\,\mathbf{I} + (\beta_2 + \beta_3\,I_1)\,\mathbf{B} + \beta_3\,I_3\,\mathbf{B}^{-1}\Big).\end{aligned} \tag{Z1.5}$$

Obliczamy kolejno współczynniki:

$$\beta_1 - \beta_3\,I_2 = \alpha_3\,I_3 - (-\alpha_2)\,I_2 = \alpha_3\,I_3 + \alpha_2\,I_2, \tag{Z1.6}$$

$$\beta_2 + \beta_3\,I_1 = (\alpha_1 + \alpha_2\,I_1) + (-\alpha_2)\,I_1 = \alpha_1, \tag{Z1.7}$$

$$\beta_3\,I_3 = -\alpha_2\,I_3 = -\alpha_2\,J^2. \tag{Z1.8}$$

Podstawiając (Z1.6)–(Z1.8) do (Z1.5):

$$\boldsymbol{\sigma} = \frac{2}{J}\Big[\Big(\alpha_2\,I_2 + \alpha_3\,I_3\Big)\,\mathbf{I} + \alpha_1\,\mathbf{B}\Big] - 2\,\alpha_2\,J\,\mathbf{B}^{-1}. \tag{Z1.9}$$

Zamiana zmiennych

Przyjmuje się nową postać funkcji energii sprężystości $W = U\big(\bar{I}_1, \bar{I}_2, J\big)$, gdzie niezmienniki deformacji izochorycznej definiuje się jako:

$$\bar{I}_1 = J^{-2/3}\,I_1 = \operatorname{tr}\bar{\mathbf{B}}, \qquad \bar{I}_2 = J^{-4/3}\,I_2 = \operatorname{tr}\bar{\mathbf{B}}^{-1}, \tag{Z1.10}$$

przy czym $\bar{\mathbf{B}} = J^{-2/3}\,\mathbf{B}$ jest lewym tensorem deformacji izochorycznej Cauchy'ego-Greena. Druga z powyższych tożsamości wynika z zapisu drugiego niezmiennika przez kofaktor. Przypominając definicje niezmienników tensora $\mathbf{B}$:

$$I_1 = \operatorname{tr}\mathbf{B}, \qquad I_2 = \tfrac{1}{2}\Big[(\operatorname{tr}\mathbf{B})^2 - \operatorname{tr}\mathbf{B}^2\Big] = \operatorname{tr}\operatorname{Cof}\mathbf{B}, \qquad I_3 = \det\mathbf{B} > 0, \tag{Z1.11}$$

kofaktor tensora odwracalnego wyraża się jako $\operatorname{Cof}\mathbf{B} = (\det\mathbf{B})\,\mathbf{B}^{-1} = I_3\,\mathbf{B}^{-1}$, skąd $I_2 = \operatorname{tr}\operatorname{Cof}\mathbf{B} = I_3\operatorname{tr}\mathbf{B}^{-1}$. Korzystając z tej relacji:

$$\bar{I}_2 = J^{-4/3}\,I_2 = J^{-4/3}\,I_3\operatorname{tr}\mathbf{B}^{-1} = J^{-4/3}\cdot J^2\operatorname{tr}\mathbf{B}^{-1} = J^{2/3}\operatorname{tr}\mathbf{B}^{-1} = \operatorname{tr}\Big(J^{2/3}\,\mathbf{B}^{-1}\Big) = \operatorname{tr}\bar{\mathbf{B}}^{-1}, \tag{Z1.12}$$

gdzie wykorzystano $\bar{\mathbf{B}}^{-1} = J^{2/3}\,\mathbf{B}^{-1}$.

Stosując regułę łańcuchową, pochodne α_k wyrażają się jako:

$$\alpha_1 = \frac{\partial U}{\partial I_1} = J^{-2/3}\,\frac{\partial U}{\partial \bar{I}_1}, \tag{Z1.13}$$

$$\alpha_2 = \frac{\partial U}{\partial I_2} = J^{-4/3}\,\frac{\partial U}{\partial \bar{I}_2}, \tag{Z1.14}$$

$$\alpha_3 = \frac{\partial U}{\partial I_3} = \frac{1}{2J}\,\frac{\partial U}{\partial J} - \frac{\bar{I}_1}{3\,J^2}\,\frac{\partial U}{\partial \bar{I}_1} - \frac{2\,\bar{I}_2}{3\,J^2}\,\frac{\partial U}{\partial \bar{I}_2}. \tag{Z1.15}$$

Podstawienie do (Z1.9)

Równanie (Z1.9) zawiera trzy składniki, które przekształcamy oddzielnie, korzystając z (Z1.13)–(Z1.15) oraz z $I_3 = J^2$:

(i) składnik z $\mathbf{I}$: $\frac{2}{J}\Big(\alpha_2\,I_2 + \alpha_3\,I_3\Big)\,\mathbf{I}$,

(ii) składnik z $\mathbf{B}$: $\frac{2}{J}\,\alpha_1\,\mathbf{B}$,

(iii) składnik z $\mathbf{B}^{-1}$: $-2\,\alpha_2\,J\,\mathbf{B}^{-1}$.

Ad (i). Korzystając z (Z1.15) i $I_3 = J^2$ mamy:

$$\alpha_3\,J^2 = \frac{J}{2}\,\frac{\partial U}{\partial J} - \frac{\bar{I}_1}{3}\,\frac{\partial U}{\partial \bar{I}_1} - \frac{2\,\bar{I}_2}{3}\,\frac{\partial U}{\partial \bar{I}_2}. \tag{Z1.16}$$

Natomiast z (Z1.14) i $I_2 = J^{4/3}\,\bar{I}_2$:

$$\alpha_2\,I_2 = J^{-4/3}\,\frac{\partial U}{\partial \bar{I}_2}\cdot J^{4/3}\,\bar{I}_2 = \bar{I}_2\,\frac{\partial U}{\partial \bar{I}_2}. \tag{Z1.17}$$

Sumując oraz upraszczając (Z1.16) i (Z1.17) otrzymujemy:

$$\alpha_2\,I_2 + \alpha_3\,I_3 = \frac{J}{2}\,\frac{\partial U}{\partial J} - \frac{\bar{I}_1}{3}\,\frac{\partial U}{\partial \bar{I}_1} + \frac{\bar{I}_2}{3}\,\frac{\partial U}{\partial \bar{I}_2}, \tag{Z1.18}$$

i cały składnik (i) wynosi:

$$\frac{2}{J}\Big(\alpha_2\,I_2 + \alpha_3\,I_3\Big)\,\mathbf{I} = \left(\frac{\partial U}{\partial J} - \frac{2\,\bar{I}_1}{3\,J}\,\frac{\partial U}{\partial \bar{I}_1} + \frac{2\,\bar{I}_2}{3\,J}\,\frac{\partial U}{\partial \bar{I}_2}\right)\mathbf{I}. \tag{Z1.19}$$

Ad (ii). Podstawiając (Z1.13) i $\mathbf{B} = J^{2/3}\,\bar{\mathbf{B}}$:

$$\frac{2}{J}\,\alpha_1\,\mathbf{B} = \frac{2}{J}\cdot J^{-2/3}\,\frac{\partial U}{\partial \bar{I}_1}\cdot J^{2/3}\,\bar{\mathbf{B}} = \frac{2}{J}\,\frac{\partial U}{\partial \bar{I}_1}\,\bar{\mathbf{B}}. \tag{Z1.20}$$

Ad (iii). Podstawiając (Z1.14) i $\mathbf{B}^{-1} = J^{-2/3}\,\bar{\mathbf{B}}^{-1}$ (wynikające z $\bar{\mathbf{B}}^{-1} = J^{2/3}\,\mathbf{B}^{-1}$):

$$-2\,\alpha_2\,J\,\mathbf{B}^{-1} = -2\cdot J^{-4/3}\,\frac{\partial U}{\partial \bar{I}_2}\cdot J\cdot J^{-2/3}\,\bar{\mathbf{B}}^{-1} = -\frac{2}{J}\,\frac{\partial U}{\partial \bar{I}_2}\,\bar{\mathbf{B}}^{-1}. \tag{Z1.21}$$

Sumując (Z1.19), (Z1.20) i (Z1.21) otrzymujemy:

$$\boldsymbol{\sigma} = \left(\frac{\partial U}{\partial J} - \frac{2\,\bar{I}_1}{3\,J}\,\frac{\partial U}{\partial \bar{I}_1} + \frac{2\,\bar{I}_2}{3\,J}\,\frac{\partial U}{\partial \bar{I}_2}\right)\mathbf{I} + \frac{2}{J}\left(\frac{\partial U}{\partial \bar{I}_1}\,\bar{\mathbf{B}} - \frac{\partial U}{\partial \bar{I}_2}\,\bar{\mathbf{B}}^{-1}\right). \tag{Z1.22}$$

Wprowadzenie dewiatorów i postać końcowa

Dewiatory lewego tensora deformacji izochorycznej Cauchy'ego-Greena definiuje się jako:

$$\bar{\mathbf{B}}_D = \bar{\mathbf{B}} - \tfrac{1}{3}\,\bar{I}_1\,\mathbf{I}, \qquad \bar{\mathbf{B}}_D^{-1} = \bar{\mathbf{B}}^{-1} - \tfrac{1}{3}\,\bar{I}_2\,\mathbf{I}, \tag{Z1.23}$$

przy czym ślady wynoszą odpowiednio $\operatorname{tr}\bar{\mathbf{B}} = \bar{I}_1$ i $\operatorname{tr}\bar{\mathbf{B}}^{-1} = \bar{I}_2$ zgodnie z (Z1.10). Odwracając (Z1.23):

$$\bar{\mathbf{B}} = \bar{\mathbf{B}}_D + \tfrac{1}{3}\,\bar{I}_1\,\mathbf{I}, \qquad \bar{\mathbf{B}}^{-1} = \bar{\mathbf{B}}_D^{-1} + \tfrac{1}{3}\,\bar{I}_2\,\mathbf{I}. \tag{Z1.24}$$

Podstawiając (Z1.24) do (Z1.22) otrzymujemy ostatecznie:

$$\boxed{\boldsymbol{\sigma} = \frac{\partial U}{\partial J}\,\mathbf{I} + \frac{2}{J}\left(\frac{\partial U}{\partial \bar{I}_1}\,\bar{\mathbf{B}}_D - \frac{\partial U}{\partial \bar{I}_2}\,\bar{\mathbf{B}}_D^{-1}\right).} \tag{Z1.25}$$

Powyższa postać jest zgodna z równaniem podanym w [67].

Z1.2. Opis Lagrange'a – II tensor naprężenia Pioli-Kirchhoffa

Punkt wyjścia

Analogiczne wyprowadzenie przeprowadza się dla naprężenia Pioli-Kirchhoffa drugiego rodzaju $\mathbf{T}$, wyrażonego przez prawy tensor deformacji Cauchy'ego-Greena $\mathbf{C} = \mathbf{F}^T\mathbf{F}$. Relacja konstytutywna przyjmuje postać:

$$\mathbf{T} = 2\left[\left(\frac{\partial W}{\partial I_1} + I_1\,\frac{\partial W}{\partial I_2}\right)\mathbf{I} - \frac{\partial W}{\partial I_2}\,\mathbf{C} + \frac{J}{2}\,\frac{\partial W}{\partial J}\,\mathbf{C}^{-1}\right]. \tag{Z1.26}$$

Podstawienie do (Z1.26)

Równanie (Z1.26) zawiera trzy składniki, które przekształcamy oddzielnie, korzystając z (Z1.13)–(Z1.15) oraz z $I_3 = J^2$:

(i) składnik z $\mathbf{I}$: $2(\alpha_1 + \alpha_2\,I_1)\,\mathbf{I}$,

(ii) składnik z $\mathbf{C}$: $-2\,\alpha_2\,\mathbf{C}$,

(iii) składnik z $\mathbf{C}^{-1}$: $J\,\frac{\partial W}{\partial J}\,\mathbf{C}^{-1} = 2\,\alpha_3\,I_3\,\mathbf{C}^{-1}$.

Ad (i). Korzystając z (Z1.13) i $I_1 = J^{2/3}\,\bar{I}_1$:

$$\alpha_1 + \alpha_2\,I_1 = J^{-2/3}\,\frac{\partial U}{\partial \bar{I}_1} + J^{-4/3}\,\frac{\partial U}{\partial \bar{I}_2}\cdot J^{2/3}\,\bar{I}_1 = J^{-2/3}\,\frac{\partial U}{\partial \bar{I}_1} + J^{-2/3}\,\bar{I}_1\,\frac{\partial U}{\partial \bar{I}_2}, \tag{Z1.27}$$

zatem cały składnik (i) wynosi:

$$2(\alpha_1 + \alpha_2\,I_1)\,\mathbf{I} = 2\,J^{-2/3}\left(\frac{\partial U}{\partial \bar{I}_1} + \bar{I}_1\,\frac{\partial U}{\partial \bar{I}_2}\right)\mathbf{I}. \tag{Z1.28}$$

Ad (ii). Podstawiając (Z1.14) i $\mathbf{C} = J^{2/3}\,\bar{\mathbf{C}}$:

$$-2\,\alpha_2\,\mathbf{C} = -2\cdot J^{-4/3}\,\frac{\partial U}{\partial \bar{I}_2}\cdot J^{2/3}\,\bar{\mathbf{C}} = -2\,J^{-2/3}\,\frac{\partial U}{\partial \bar{I}_2}\,\bar{\mathbf{C}}. \tag{Z1.29}$$

Ad (iii). Trzeci składnik z (Z1.26) zapisujemy korzystając z (Z1.15) i $I_3 = J^2$:

$$\alpha_3\,I_3 = \frac{J}{2}\,\frac{\partial U}{\partial J} - \frac{\bar{I}_1}{3}\,\frac{\partial U}{\partial \bar{I}_1} - \frac{2\,\bar{I}_2}{3}\,\frac{\partial U}{\partial \bar{I}_2}, \tag{Z1.30}$$

co daje (z $\mathbf{C}^{-1} = J^{-2/3}\,\bar{\mathbf{C}}^{-1}$, wynikającym z $\bar{\mathbf{C}} = J^{-2/3}\,\mathbf{C}$):

$$2\,\alpha_3\,I_3\,\mathbf{C}^{-1} = 2\left(\frac{J}{2}\,\frac{\partial U}{\partial J} - \frac{\bar{I}_1}{3}\,\frac{\partial U}{\partial \bar{I}_1} - \frac{2\,\bar{I}_2}{3}\,\frac{\partial U}{\partial \bar{I}_2}\right)J^{-2/3}\,\bar{\mathbf{C}}^{-1}. \tag{Z1.31}$$

Sumując (Z1.28), (Z1.29) i (Z1.31), grupując składniki przy $\bar{\mathbf{C}}^{-1}$ oraz łącząc wyrażenia przy $\mathbf{I}$ z wkładami z (iii), otrzymujemy:

$$\mathbf{T} = J^{-2/3}\left[2\left(\frac{\partial U}{\partial \bar{I}_1} + 2\,\bar{I}_1\,\frac{\partial U}{\partial \bar{I}_2}\right)\mathbf{I} - 2\,\frac{\partial U}{\partial \bar{I}_2}\,\bar{\mathbf{C}}\right] + 2\,J^{-2/3}\left(\frac{J}{2}\,\frac{\partial U}{\partial J} - \frac{\bar{I}_1}{3}\,\frac{\partial U}{\partial \bar{I}_1} - \frac{2\,\bar{I}_2}{3}\,\frac{\partial U}{\partial \bar{I}_2}\right)\bar{\mathbf{C}}^{-1}. \tag{Z1.32}$$

Wprowadzenie dewiatorów

Ponieważ $\mathbf{T}$ jest tensorem w konfiguracji Lagrange'a, dewiator dowolnego tensora $\mathbf{A}$ w tej konfiguracji definiuje się jako:

$$\mathrm{DEV}\,\mathbf{A} = \mathbf{A} - \tfrac{1}{3}\mathrm{tr}(\mathbf{A}\,\mathbf{C})\,\mathbf{C}^{-1}, \tag{Z1.33}$$

Analogicznie w zmiennych izochorycznych, korzystając z $\mathbf{C} = J^{2/3}\,\bar{\mathbf{C}}$ i $\mathbf{C}^{-1} = J^{-2/3}\,\bar{\mathbf{C}}^{-1}$:

$$\mathrm{DEV}\,\mathbf{A} = \mathbf{A} - \tfrac{1}{3}\mathrm{tr}\Big(\mathbf{A}\,\bar{\mathbf{C}}\Big)\,\bar{\mathbf{C}}^{-1}. \tag{Z1.34}$$

Korzystając z definicji (Z1.34):

$$\mathrm{DEV}\,\mathbf{I} = \mathbf{I} - \tfrac{1}{3}\,\mathrm{tr}(\bar{\mathbf{C}})\,\bar{\mathbf{C}}^{-1} = \mathbf{I} - \tfrac{1}{3}\,\bar{I}_1\,\bar{\mathbf{C}}^{-1} \;\rightarrow\; \mathbf{I} = \mathrm{DEV}\,\mathbf{I} + \tfrac{1}{3}\,\bar{I}_1\,\bar{\mathbf{C}}^{-1}, \tag{Z1.35}$$

$$\mathrm{DEV}\,\bar{\mathbf{C}} = \bar{\mathbf{C}} - \tfrac{1}{3}\,\mathrm{tr}(\bar{\mathbf{C}}^2)\,\bar{\mathbf{C}}^{-1} = \bar{\mathbf{C}} - \tfrac{1}{3}\Big(\bar{I}_1^2 - 2\,\bar{I}_2\Big)\,\bar{\mathbf{C}}^{-1} \;\rightarrow\; \bar{\mathbf{C}} = \mathrm{DEV}\,\bar{\mathbf{C}} + \tfrac{1}{3}\Big(\bar{I}_1^2 - 2\,\bar{I}_2\Big)\,\bar{\mathbf{C}}^{-1}. \tag{Z1.36}$$

Podstawiając (Z1.35) i (Z1.36) do pierwszego wiersza (Z1.32):

$$
\begin{aligned}
&J^{-2/3}\left[2\left(\frac{\partial U}{\partial \bar{I}_1}+\bar{I}_1\,\frac{\partial U}{\partial \bar{I}_2}\right)\mathbf{I}-2\,\frac{\partial U}{\partial \bar{I}_2}\,\bar{\mathbf{C}}\right]\\
&=J^{-2/3}\left[2\left(\frac{\partial U}{\partial \bar{I}_1}+\bar{I}_1\,\frac{\partial U}{\partial \bar{I}_2}\right)\left(\mathrm{DEV}\,\mathbf{I}+\tfrac{1}{3}\,\bar{I}_1\,\bar{\mathbf{C}}^{-1}\right)-2\,\frac{\partial U}{\partial \bar{I}_2}\left(\mathrm{DEV}\,\bar{\mathbf{C}}+\tfrac{1}{3}\left(\bar{I}_1^2-2\,\bar{I}_2\right)\bar{\mathbf{C}}^{-1}\right)\right]\\
&=J^{-2/3}\left[2\left(\frac{\partial U}{\partial \bar{I}_1}+\bar{I}_1\,\frac{\partial U}{\partial \bar{I}_2}\right)\mathrm{DEV}\,\mathbf{I}-2\,\frac{\partial U}{\partial \bar{I}_2}\,\mathrm{DEV}\,\bar{\mathbf{C}}\right]\\
&\quad+\frac{2}{3}\,J^{-2/3}\left[\bar{I}_1\,\frac{\partial U}{\partial \bar{I}_1}+\bar{I}_1^2\,\frac{\partial U}{\partial \bar{I}_2}-\left(\bar{I}_1^2-2\,\bar{I}_2\right)\frac{\partial U}{\partial \bar{I}_2}\right]\bar{\mathbf{C}}^{-1}\\
&=J^{-2/3}\left[2\left(\frac{\partial U}{\partial \bar{I}_1}+\bar{I}_1\,\frac{\partial U}{\partial \bar{I}_2}\right)\mathrm{DEV}\,\mathbf{I}-2\,\frac{\partial U}{\partial \bar{I}_2}\,\mathrm{DEV}\,\bar{\mathbf{C}}\right]+\frac{2}{3}\,J^{-2/3}\left[\bar{I}_1\,\frac{\partial U}{\partial \bar{I}_1}+2\,\bar{I}_2\,\frac{\partial U}{\partial \bar{I}_2}\right]\bar{\mathbf{C}}^{-1}.
\end{aligned}
\tag{Z1.37}
$$

Dodając drugi wiersz (Z1.32), zbieramy wyrazy przy $\bar{\mathbf{C}}^{-1}$:

$$
\frac{2}{3}\,J^{-2/3}\left[\bar{I}_1\,\frac{\partial U}{\partial \bar{I}_1}+2\,\bar{I}_2\,\frac{\partial U}{\partial \bar{I}_2}\right]+2\,J^{-2/3}\left[\frac{J}{2}\,\frac{\partial U}{\partial J}-\frac{\bar{I}_1}{3}\,\frac{\partial U}{\partial \bar{I}_1}-\frac{2\,\bar{I}_2}{3}\,\frac{\partial U}{\partial \bar{I}_2}\right]=J^{1/3}\,\frac{\partial U}{\partial J},
\tag{Z1.38}
$$

ponieważ składniki z $\partial U/\partial \bar{I}_1$ i $\partial U/\partial \bar{I}_2$ znoszą się parami.

Ostatecznie, korzystając z liniowości operatora DEV otrzymujemy:

$$
\boxed{\mathbf{T}=J^{1/3}\,\frac{\partial U}{\partial J}\,\bar{\mathbf{C}}^{-1}+J^{-2/3}\,\mathrm{DEV}\left(2\left(\frac{\partial U}{\partial \bar{I}_1}+\bar{I}_1\,\frac{\partial U}{\partial \bar{I}_2}\right)\mathbf{I}-2\,\frac{\partial U}{\partial \bar{I}_2}\,\bar{\mathbf{C}}\right).}
\tag{Z1.39}
$$

Korzystając z $\bar{\mathbf{C}}^{-1}=J^{2/3}\,\mathbf{C}^{-1}$, równanie (Z1.39) przyjmuje równoważną postać zgodną z [64]:

$$
\mathbf{T}=J\,\frac{\partial U}{\partial J}\,\mathbf{C}^{-1}+J^{-2/3}\,\mathrm{DEV}\left(2\left(\frac{\partial U}{\partial \bar{I}_1}+\bar{I}_1\,\frac{\partial U}{\partial \bar{I}_2}\right)\mathbf{I}-2\,\frac{\partial U}{\partial \bar{I}_2}\,\bar{\mathbf{C}}\right).
\tag{Z1.40}
$$

Załącznik 2. Tabela porównawcza relacji konstytutywnych hipersprężystości

Tabela Z2.1. Porównanie postaci funkcji energii sprężystości oraz relacji konstytutywnych dla izotropowych materiałów hipersprężystych

Typ materiału	**Funkcja ES**	**Tensor naprężenia Pioli-Kirchhoffa II rodzaju T**	**Tensor naprężenia Cauchy'ego $\boldsymbol{\sigma}$**
Ściśliwy (postać ogólna)	$W = W(I_1, I_2, I_3)$	$\mathbf{T} = 2\big(\gamma_1\,\mathbf{I} + \gamma_2\,\mathbf{C} + \gamma_3\,\mathbf{C}^2\big)$ gdzie: $\gamma_1 = \alpha_1 + \alpha_2\,I_1 + \alpha_3\,I_2, \quad \gamma_2 = -(\alpha_2 + \alpha_3\,I_1), \quad \gamma_3 = \alpha_3.$ *Postać alternatywna:* $\mathbf{T} = 2\left[(\alpha_1 + \alpha_2 I_1)\mathbf{I} - \alpha_2\mathbf{C} + \alpha_3 I_3 \mathbf{C}^{-1}\right]$	$\boldsymbol{\sigma} = \frac{2}{J}\big(\beta_1\,\mathbf{I} + \beta_2\,\mathbf{B} + \beta_3\,\mathbf{B}^2\big)$ gdzie: $\beta_1 = \alpha_3\,I_3, \quad \beta_2 = \alpha_1 + \alpha_2\,I_1, \quad \beta_3 = -\alpha_2,$ *Postać alternatywna:* $\boldsymbol{\sigma} = \frac{2}{J}\Big[\big(\alpha_2\,I_2 + \alpha_3\,I_3\big)\,\mathbf{I} + \alpha_1\,\mathbf{B}\Big] - 2\,\alpha_2\,J\,\mathbf{B}^{-1}$
Ściśliwy (postać ogólna z niezmiennikami izochorycznymi i J)	$W = U(\bar{I}_1, \bar{I}_2, J)$	$\mathbf{T} = J^{1/3}\,\frac{\partial U}{\partial J}\,\bar{\mathbf{C}}^{-1} + J^{-2/3}\,\mathrm{DEV}\Big(2\Big(\frac{\partial U}{\partial \bar{I}_1} + \bar{I}_1\,\frac{\partial U}{\partial \bar{I}_2}\Big)\,\mathbf{I} - 2\,\frac{\partial U}{\partial \bar{I}_2}\,\bar{\mathbf{C}}\Big).$	$\boldsymbol{\sigma} = \frac{\partial U}{\partial J}\,\mathbf{I} + \frac{2}{J}\left(\frac{\partial U}{\partial \bar{I}_1}\,\bar{\mathbf{B}}_D - \frac{\partial U}{\partial \bar{I}_2}\,\bar{\mathbf{B}}_D^{-1}\right)$
Małościśliwy	$W = W_D(\bar{I}_1, \bar{I}_2) + W_I(J)$ W_D – cz. izochoryczna, W_I – cz. objętościowa	$\mathbf{T} = J^{1/3}\,\frac{\partial W_I}{\partial J}\,\bar{\mathbf{C}}^{-1} + J^{-2/3}\,\mathrm{DEV}\Big(2\Big(\frac{\partial W_D}{\partial \bar{I}_1} + \bar{I}_1\,\frac{\partial W_D}{\partial \bar{I}_2}\Big)\,\mathbf{I} - 2\,\frac{\partial W_D}{\partial \bar{I}_2}\,\bar{\mathbf{C}}\Big).$	$\boldsymbol{\sigma} = \frac{\partial W_I}{\partial J}\,\mathbf{I} + \frac{2}{J}\left(\frac{\partial W_D}{\partial \bar{I}_1}\,\bar{\mathbf{B}}_D - \frac{\partial W_D}{\partial \bar{I}_2}\,\bar{\mathbf{B}}_D^{-1}\right)$
Nieściśliwy	$W = W_D(\bar{I}_1, \bar{I}_2)$ $(\det J = 1)$	$\mathbf{T} = -p\,\bar{\mathbf{C}}^{-1} + 2\left(\frac{\partial W_D}{\partial \bar{I}_1} + \bar{I}_1\frac{\partial W_D}{\partial \bar{I}_2}\right)\mathbf{I} - 2\,\frac{\partial W_D}{\partial \bar{I}_2}\,\bar{\mathbf{C}}$	$\boldsymbol{\sigma} = -p\mathbf{I} + 2\frac{\partial W_D}{\partial \bar{I}_1}\bar{\mathbf{B}} - 2\frac{\partial W_D}{\partial \bar{I}_2}\bar{\mathbf{B}}^{-1}$

Oznaczenia: W – funkcja jednostkowej energii sprężystości (JES), $\mathbf{C} = \mathbf{F}^T\mathbf{F}$ – prawy tensor deformacji Cauchy'ego–Greena, $\mathbf{B} = \mathbf{F}\mathbf{F}^T$ – lewy tensor deformacji, $J = \det\mathbf{F}$, $I_1 = \mathrm{tr}\,\mathbf{C}$, $I_2 = \mathrm{tr}(\mathrm{Cof}\,\mathbf{C})$, $I_3 = \det\mathbf{C} = J^2$, $\bar{I}_k = J^{-2k/3}I_k$ – niezmienniki izochoryczne $(k = 1, 2)$, $\bar{\mathbf{C}} = J^{-2/3}\mathbf{C}$ – izochoryczny prawy tensor deformacji, $\bar{\mathbf{B}} = J^{-2/3}\mathbf{B}$ – izochoryczny lewy tensor deformacji, $\bar{\mathbf{B}}_D = \mathrm{dev}(\bar{\mathbf{B}})$ – dewiator tensora $\bar{\mathbf{B}}$, $\bar{\mathbf{B}}_D^{-1} = \mathrm{dev}(\bar{\mathbf{B}}^{-1})$, $\mathrm{DEV}(\bullet) = (\bullet) - \frac{1}{3}\,\mathrm{tr}[(\bullet)\,\mathbf{C}]\,\mathbf{C}^{-1}$ – operator dewiatora materialnego (w konfiguracji odniesienia), $\mathrm{dev}(\bullet) = (\bullet) - \frac{1}{3}\,\mathrm{tr}(\bullet)\,\mathbf{I}$ – operator dewiatora w konfiguracji materialnej, $\alpha_i = \partial W/\partial I_i$ $(i = 1, 2, 3)$, p – mnożnik Lagrange'a wymuszający warunek nieściśliwości.

```
class TensorToInvariantsTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, compressibility="compressible",
    ↪  incompressible_method="lagrange"):
        self.compressibility = compressibility
        self.incompressible_method = incompressible_method
        self.decoupled = is_decoupled(compressibility)

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        F = X.clone()
        J = cm.invariant_I3(F)
        if self.decoupled:
            F = J.pow(-1/3).view(-1, 1, 1) * F
        C = cm.ten2_C(F)
        I1 = cm.invariant_I1(C)
        I2 = cm.invariant_I2(C)
        invariants = np.stack([I1.numpy(), I2.numpy(), J.numpy()], axis=1)
        return {'F': F.numpy(), 'C': C.numpy(), 'inv': invariants}

class DesignMatrixTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, degree=2, model_fit='simplified', model_type=None,
    ↪  stress_free=True,
             convexity=False, compressibility='compressible',
             ↪  incompressible_method='lagrange',
             n_exp=1, exp_max_arg=5.0):
        self.degree = degree
        self.model_fit = model_fit
        self.model_type = model_type
        self.decoupled = is_decoupled(compressibility)
        self.stress_free = stress_free
        self.convexity = convexity
        self.compressibility = compressibility
        self.incompressible_method = incompressible_method
        self.n_exp = n_exp
        self.exp_max_arg = exp_max_arg
```

```
        self.I1, self.I2, self.J = sp.symbols('I1 I2 J')

        self.feature_syms = None

        self.dpsi_dI1_syms = None
        self.dpsi_dI2_syms = None
        self.dpsi_dJ_syms = None

        self.dpsi_dI1_funcs = None
        self.dpsi_dI2_funcs = None
        self.dpsi_dJ_funcs = None

        self.hessian_funcs = None

        self.feature_names = None
        self.feature_syms = None

    def fit(self, X, y=None):
        inv = X['inv']

        if self.model_type == 'reduced-hartmann':
            I1_arg_max = inv[:, 0].max() - 3
            I2_arg_max = inv[:, 1].max()**1.5 - 3 * np.sqrt(3)

            alpha_I1 = self.exp_max_arg / I1_arg_max
            alpha_I2 = self.exp_max_arg / I2_arg_max
        else:
            alpha_I1 = alpha_I2 = None

        fg = FeatureGenerator(
            invariants=inv,
            poly=None,
            model_type=self.model_type,
            stress_free=self.stress_free,
            compressibility=self.compressibility,
            incompressible_method=self.incompressible_method,
            N=self.degree,
            n_exp=self.n_exp,
```

```
            alpha_I1=alpha_I1,
            alpha_I2=alpha_I2,
        )
        self.fg_ = fg
        self.feature_names = fg.get_feature_names_out()
        self.feature_syms = fg.get_feature_symbols()

        self.feature_funcs_ = [sp.lambdify((self.I1, self.I2, self.J), s, 'numpy')
                               for s in self.feature_syms]
        self.dpsi_dI1_funcs = [sp.lambdify((self.I1, self.I2, self.J), sp.diff(s,
        ↪  self.I1), 'numpy')
                               for s in self.feature_syms]
        self.dpsi_dI2_funcs = [sp.lambdify((self.I1, self.I2, self.J), sp.diff(s,
        ↪  self.I2), 'numpy')
                               for s in self.feature_syms]
        self.dpsi_dJ_funcs  = [sp.lambdify((self.I1, self.I2, self.J), sp.diff(s,
        ↪  self.J),  'numpy')
                               for s in self.feature_syms]
        if self.convexity:
            self.hessian_funcs = [sp.lambdify((self.I1, self.I2, self.J),
            ↪  sp.hessian(s, (self.I1, self.I2, self.J)), 'numpy')
                                  for s in self.feature_syms]
        return self


    def transform(self, X):

        F = X['F']
        C = X['C']
        invariants = X['inv']
        I1 = invariants[:, 0]
        I2 = invariants[:, 1]
        J = invariants[:, 2]

        dpsi_dI1 = np.array([
            [f(I1_i, I2_i, J_i) for f in self.dpsi_dI1_funcs]
            for I1_i, I2_i, J_i in zip(I1, I2, J)
        ])
```

```
dpsi_dI2 = np.array([
    [f(I1_i, I2_i, J_i) for f in self.dpsi_dI2_funcs]
    for I1_i, I2_i, J_i in zip(I1, I2, J)
])

dpsi_dJ = np.array([
    [f(I1_i, I2_i, J_i) for f in self.dpsi_dJ_funcs]
    for I1_i, I2_i, J_i in zip(I1, I2, J)
])

design_rows = []
for idx in range(len(I1)):
    row = self._compute_S_terms(
        F[idx],
        C[idx],
        I1[idx],
        J[idx],
        dpsi_dI1[idx],
        dpsi_dI2[idx],
        dpsi_dJ[idx],
    )
    design_rows.append(row)
design_matrix = np.vstack(design_rows)

if self.convexity:
    identity_F = torch.eye(3).unsqueeze(0)
    identity_C = cm.ten2_C(identity_F)
    identity_C_np = identity_C.numpy()

    I1_0 = cm.invariant_I1(identity_C).item()
    I2_0 = cm.invariant_I2(identity_C).item()
    J_0 = cm.invariant_I3(identity_F).item()

    dpsi_dI1_0 = np.array([f(I1_0, I2_0, J_0) for f in self.dpsi_dI1_funcs])
    dpsi_dI2_0 = np.array([f(I1_0, I2_0, J_0) for f in self.dpsi_dI2_funcs])
    dpsi_dJ_0 = np.array([f(I1_0, I2_0, J_0) for f in self.dpsi_dJ_funcs])

    identity_F_np = identity_F.squeeze(0).numpy()
```

```
        design_matrix_0 = self._compute_S_terms(identity_F_np, identity_C_np,
        ↪  I1_0, J_0, dpsi_dI1_0, dpsi_dI2_0, dpsi_dJ_0)
        return {
            'X': design_matrix,
            'design_matrix_zero': design_matrix_0,
            'dpsi_dI1_funcs': self.dpsi_dI1_funcs,
            'dpsi_dI2_funcs': self.dpsi_dI2_funcs,
            'dpsi_dJ_funcs': self.dpsi_dJ_funcs,
            'invariants': invariants,
            'hessian_funcs': self.hessian_funcs
        }
    else:
        return design_matrix

def _compute_S_terms(self, F_i, C_i, I1_i, J_i, dpsi_dI1_i, dpsi_dI2_i,
↪  dpsi_dJ_i):

    C_inv = np.linalg.inv(C_i)

    term1 = (dpsi_dI1_i + I1_i * dpsi_dI2_i)[:, None, None] * np.eye(3)
    term2 = -dpsi_dI2_i[:, None, None] * C_i

    if self.decoupled:
        if self.compressibility == "slightly_compressible":
            T_iso = 2 * J_i**(-2/3) * cm.deviatoric_operator(term1 + term2,
            ↪  C_i).numpy()
            term3 = (J_i**(1/3) * dpsi_dJ_i)[:, None, None] * C_inv
            T = T_iso + term3
        else:
            T_iso = 2 * (term1 + term2)
            if self.compressibility == "incompressible" and
            ↪  self.incompressible_method == "lagrange":
                C11, C22 = C_i[0, 0], C_i[1, 1]
                avg = 0.5 * (C11 + C22)
                if avg > 1e-12 and abs(C11 - C22) / avg < 1e-3:
                    side = 2
                else:
                    side = 1
```

```
            p = T_iso[:, side, side] / C_inv[side, side]
            T = T_iso - p[:, None, None] * C_inv
        elif self.compressibility == "incompressible" and
        ↪  self.incompressible_method == "penalty":
            raise NotImplementedError(
                "Penalty formulation for incompressible material is not
                ↪  implemented yet. "
                "Use incompressible_method='lagrange' for Treloar data."
            )

else:
    term3 = (0.5 * J_i * dpsi_dJ_i)[:, None, None] * C_inv
    T = 2 * (term1 + term2 + term3)

F_for_stress = F_i

if self.decoupled:
    F_for_stress = J_i**(1/3) * F_i

S = np.einsum('ij,kjl->kil', F_for_stress, T)

if self.model_fit == "simplified":
    return S[:, 0, 0]
elif self.model_fit == "full":
    return S.reshape(S.shape[0], -1)
else:
    raise ValueError(f"Unknown model_fit: {self.model_fit}")
```

Załącznik 4. Generowanie funkcji bazowych

Listing 1. Wydruk z klasy FeatureGenerator dla stopnia wielomianu $n = 2$

```
=== Test: decoupled=rivlin, stress-free reference configuration=False ===
Sklearn feature names: (I1-3), (I2-3), (I1-3)^2, (I1-3) (I2-3), (I2-3)^2, (J-1)^2

=== Test: decoupled=rivlin, stress-free reference configuration=True ===
Sklearn feature names: (I1-3), (I2-3), (I1-3)^2, (I1-3) (I2-3), (I2-3)^2, (J-1)^2

=== Test: decoupled=silhavy, stress-free reference configuration=False ===
Sklearn feature names: (I1-3), (I2-3), (J-1), (I1-3)^2, (I2-3)^2, (J-1)^2

=== Test: decoupled=silhavy, stress-free reference configuration=True ===
Sklearn feature names: (I1-3), (I2-3), (I1-3)^2, (I2-3)^2, (J-1)^2

=== Test: decoupled=True, stress-free reference configuration=False ===
Sklearn feature names: (I1-3), (I2-3), (J-1), (I1-3)^2, (I1-3) (I2-3), (I2-3)^2, (J-1)^2

=== Test: decoupled=True, stress-free reference configuration=True ===
Sklearn feature names: (I1-3), (I2-3), (I1-3)^2, (I1-3) (I2-3), (I2-3)^2, (J-1)^2

=== Test: decoupled=False, stress-free reference configuration=False ===
Sklearn feature names: (I1-3), (I2-3), (J-1), (I1-3)^2, (I1-3) (I2-3), (I1-3) (J-1), (I2-3)^2, (I2-3) (J-1), (J-1)^2

=== Test: decoupled=False, stress-free reference configuration=True ===
Sklearn feature names: (I1-3), (I2-3), (I1-3)^2, (I1-3) (I2-3), (I1-3) (J-1), (I2-3)^2, (I2-3) (J-1), (J-1)^2
```

Załącznik 5. Studium przypadku klasy `DesignMatrixTransformer`

Metoda `transform`

W celu rozjaśnienia działania klasy `DesignMatrixTransformer` w niniejszym załączniku przedstawiono przykładowe obliczenia numeryczne dla stopnia wielomianu funkcji jednostkowej energii sprężystości równego 2 i założenia materiału nieściśliwego będącego w stanie jednoosiowego rozciągania.

Dane wejściowe

Przykładowe dane wejściowe obejmują reprezentację numeryczną tensora gradientu deformacji izochorycznej $\bar{\mathbf{F}}$ oraz powiązaną reprezentację prawego tensora deformacji izochorycznej Cauchy'ego-Green'a $\bar{\mathbf{C}}$ wraz z wartościami pierwszych dwóch niezmienników deformacji izochorycznej $\bar{I}_1, \bar{I}_2$ dla jednej próbki:

$$\bar{\mathbf{F}} \to \begin{pmatrix} 3.016 & 0 & 0 \\ 0 & 0.576 & 0 \\ 0 & 0 & 0.576 \end{pmatrix}, \quad \bar{\mathbf{C}} \to \begin{pmatrix} 9.096 & 0 & 0 \\ 0 & 0.332 & 0 \\ 0 & 0 & 0.332 \end{pmatrix}, \quad \bar{I}_1 = 9.760,\ \bar{I}_2 = 6.142. \tag{Z5.1}$$

Metoda `fit`

W metodzie `fit` wygenerowano listę symbolicznych funkcji bazowych φ_α zbudowanych z przesuniętych niezmienników $(\bar{I}_1 - 3)$ i $(\bar{I}_2 - 3)$:

$$\text{feature\_syms} = \Big\{ (\bar{I}_1 - 3),\ (\bar{I}_2 - 3),\ (\bar{I}_1 - 3)^2,\ (\bar{I}_1 - 3)(\bar{I}_2 - 3),\ (\bar{I}_2 - 3)^2 \Big\}. \tag{Z5.2}$$

Na podstawie tej listy obliczane są pochodne cząstkowe względem $\bar{I}_1$ i $\bar{I}_2$:

$$\begin{aligned} \text{dpsi\_dI1\_syms} &= \Big\{ 1,\ 0,\ 2(\bar{I}_1 - 3),\ (\bar{I}_2 - 3),\ 0 \Big\}, \\ \text{dpsi\_dI2\_syms} &= \Big\{ 0,\ 1,\ 0,\ (\bar{I}_1 - 3),\ 2(\bar{I}_2 - 3) \Big\}. \end{aligned} \tag{Z5.3}$$

Metoda `transform`

Korzystając z funkcji numerycznych utworzonych w metodzie `fit`, dla zadanej pary $\bar{I}_1, \bar{I}_2$ obliczane są wartości numeryczne pochodnych:

$$\begin{aligned} \text{dpsi\_dI1} &= \Big\{\, 1.00,\ 0.00,\ 13.52,\ 3.14,\ 0.00 \,\Big\}, \\ \text{dpsi\_dI2} &= \Big\{\, 0.00,\ 1.00,\ 0.00,\ 6.76,\ 6.28 \,\Big\}. \end{aligned} \tag{Z5.4}$$

Wzór na tensor cech w mierze Pioli-Kirchhoffa II rodzaju dla materiału nieściśliwego przyjmuje postać:

$$\bar{\mathbf{T}}_\alpha = -\,p_\alpha\,\bar{\mathbf{C}}^{-1} + 2\left(\frac{\partial W_D}{\partial \bar{I}_1} + \bar{I}_1\,\frac{\partial W_D}{\partial \bar{I}_2}\right)\mathbf{I} - 2\,\frac{\partial W_D}{\partial \bar{I}_2}\,\bar{\mathbf{C}}, \tag{Z5.5}$$

gdzie $\mathbf{I}$ jest tensorem jednostkowym, a mnożnik p_α wyznaczany jest niezależnie dla każdej funkcji bazowej z warunku zerowania bocznej składowej naprężenia. Składowa $(\bar{T}_{11})$ dla kolejnych funkcji bazowych wynosi:

$\bar{\mathbf{T}}_\alpha \rightarrow$

	$(\bar{I}_1{-}3)$	$(\bar{I}_2{-}3)$	$(\bar{I}_1{-}3)^2$	$(\bar{I}_1{-}3)(\bar{I}_2{-}3)$	$(\bar{I}_2{-}3)^2$
$\bar{T}_{11}$	1.93	0.64	26.05	10.37	4.02
$\bar{T}_{22}$	0.00	0.00	0.00	0.00	0.00
$\bar{T}_{33}$	0.00	0.00	0.00	0.00	0.00

. (Z5.6)

Zgodnie z relacją $\bar{\mathbf{S}} = \bar{\mathbf{F}}\,\bar{\mathbf{T}}$ tensory cech są odwzorowywane z miary Pioli-Kirchhoffa II rodzaju na miarę Pioli-Kirchhoffa I rodzaju:

$$\bar{\mathbf{S}}_\alpha = \bar{\mathbf{F}}\,\bar{\mathbf{T}}_\alpha, \qquad \text{tj.} \quad \bar{S}_{11} = \bar{F}_{11}\,\bar{T}_{11},\ \ \bar{S}_{22} = \bar{F}_{22}\,\bar{T}_{22},\ \ \bar{S}_{33} = \bar{F}_{33}\,\bar{T}_{33}. \tag{Z5.7}$$

Ponieważ boczne składowe tensora cech PK2 w przypadku jednoosiowego rozciągania są zerowe, po odwzorowaniu niezerowa pozostaje wyłącznie składowa $\bar{S}_{11}$. Wartości numeryczne macierzy cech naprężeniowych wynoszą:

design_stress_matrix $\rightarrow$

	$(\bar{I}_1{-}3)$	$(\bar{I}_2{-}3)$	$(\bar{I}_1{-}3)^2$	$(\bar{I}_1{-}3)(\bar{I}_2{-}3)$	$(\bar{I}_2{-}3)^2$
$\bar{S}_{11}$	5.81	1.93	78.60	31.30	12.11

. (Z5.8)

Załącznik 6. Implementacja normalizacji cech i odtwarzania wag fizycznych w modelu CANN

models.py

```
import torch
import torch.nn as nn


def activation_exp(x: torch.Tensor) -> torch.Tensor:
    return torch.exp(x) - 1.0


class SingleInvariantBlock(nn.Module):

    def __init__(self, scale: float = 1.0) -> None:
        super().__init__()
        self.register_buffer('scale', torch.tensor(float(scale)))

        self.w_lin_x  = nn.Parameter(torch.empty(1).normal_(std=0.3).abs_())
        self.w_exp_x  = nn.Parameter(torch.empty(1).uniform_(0.0, 1e-3))
        self.w_lin_x2 = nn.Parameter(torch.empty(1).normal_(std=0.3).abs_())
        self.w_exp_x2 = nn.Parameter(torch.empty(1).uniform_(0.0, 1e-3))
        self.w_lin_x3 = nn.Parameter(torch.empty(1).normal_(std=0.3).abs_())
        self.w_exp_x3 = nn.Parameter(torch.empty(1).uniform_(0.0, 1e-3))

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        u  = x / self.scale
        u2 = u * u
        u3 = u2 * u
        return torch.cat([
            self.w_lin_x  * u,
            activation_exp(self.w_exp_x  * u),
            self.w_lin_x2 * u2,
            activation_exp(self.w_exp_x2 * u2),
            self.w_lin_x3 * u3,
            activation_exp(self.w_exp_x3 * u3),
        ], dim=1)
```

```
    def physical_weights(self) -> torch.Tensor:
        s = self.scale
        return torch.stack([
            (self.w_lin_x  / s).squeeze(),
            (self.w_exp_x  / s).squeeze(),
            (self.w_lin_x2 / s ** 2).squeeze(),
            (self.w_exp_x2 / s ** 2).squeeze(),
            (self.w_lin_x3 / s ** 3).squeeze(),
            (self.w_exp_x3 / s ** 3).squeeze(),
        ])


class StrainEnergyCANN(nn.Module):

    NUM_TERMS = 12

    def __init__(self, scale_I1: float = 1.0, scale_I2: float = 1.0) -> None:
        super().__init__()
        self.block_I1 = SingleInvariantBlock(scale_I1)
        self.block_I2 = SingleInvariantBlock(scale_I2)
        self.mix = nn.Parameter(torch.empty(self.NUM_TERMS).normal_(std=0.3).abs_())

    @classmethod
    def from_invariants(cls, I1: torch.Tensor, I2: torch.Tensor,
                        eps: float = 1e-6) -> "StrainEnergyCANN":
        s1 = (torch.as_tensor(I1, dtype=torch.float32) -
        ↪  3.0).abs().max().clamp_min(eps).item()
        s2 = (torch.as_tensor(I2, dtype=torch.float32) -
        ↪  3.0).abs().max().clamp_min(eps).item()
        return cls(scale_I1=s1, scale_I2=s2)

    def forward(self, I1: torch.Tensor, I2: torch.Tensor) -> torch.Tensor:
        feats = torch.cat([
            self.block_I1(I1 - 3.0),
            self.block_I2(I2 - 3.0),
        ], dim=1)
        return feats @ self.mix.unsqueeze(1)

    @torch.no_grad()
```

```
def clamp_nonneg(self) -> None:
    for p in self.parameters():
        p.clamp_(min=0.0)

def l2_regularization(self, weight: float = 1e-3) -> torch.Tensor:
    return weight * sum((p * p).sum() for p in self.parameters())

def weight_summary(self) -> torch.Tensor:
    inner = torch.cat([
        self.block_I1.physical_weights(),
        self.block_I2.physical_weights(),
    ])
    return torch.stack([inner, self.mix], dim=1).detach()
```

Załącznik 7. Porównanie implementacji sieci PINN: własnej w PyTorch z biblioteką DeepXDE

Listing 1. DeepXDE

```python
import deepxde as dde
import matplotlib.pyplot as plt
import numpy as np
import torch

def pde(x, y):
    dy_xx = dde.grad.hessian(y, x)
    return -dy_xx - np.pi ** 2 * torch.sin(np.pi * x)

def boundary_left(x, on_boundary):
    return on_boundary and np.isclose(x[0], -1.0)

def boundary_right(x, on_boundary):
    return on_boundary and np.isclose(x[0], 1.0)

def func(x):
    return np.sin(np.pi * x)

geom = dde.geometry.Interval(-1, 1)

bc_left = dde.icbc.DirichletBC(geom, lambda x: 0.0, boundary_left)
bc_right = dde.icbc.DirichletBC(geom, lambda x: 0.0, boundary_right)

data = dde.data.PDE(geom, pde, [bc_left, bc_right], 16, 2, solution=func, num_test=100)

layer_size = [1] + [50] * 3 + [1]
net = dde.nn.FNN(layer_size, "tanh", "Glorot uniform")

model = dde.Model(data, net)
model.compile("adam", lr=0.001, metrics=["MSE"])
losshistory, train_state = model.train(iterations=10000)

```

Listing 2. Czysty PyTorch (część 1/2)

```python
import math
import numpy as np
import torch
import torch.nn as nn
import matplotlib.pyplot as plt

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

class PINN(nn.Module):
    def __init__(self):
        super().__init__()
        self.dense1 = nn.Linear(1, 50)
        self.dense2 = nn.Linear(50, 50)
        self.dense3 = nn.Linear(50, 50)
        self.out    = nn.Linear(50, 1)
        self.act    = nn.Tanh()
        self._init_weights()

    def _init_weights(self):
        for layer in [self.dense1, self.dense2, self.dense3, self.out]:
            nn.init.xavier_uniform_(layer.weight)
            nn.init.zeros_(layer.bias)

    def forward(self, x):
        x = self.act(self.dense1(x))
        x = self.act(self.dense2(x))
        x = self.act(self.dense3(x))
        x = self.out(x)
        return x

def pde(model, x):
    x = x.requires_grad_(True)
    y = model(x)
    y_x = torch.autograd.grad(y, x, torch.ones_like(y), create_graph=True)[0]
    y_xx = torch.autograd.grad(y_x, x, torch.ones_like(y_x), create_graph=True)[0]
    return y_xx + (math.pi**2) * torch.sin(math.pi * x)
```

Listing 2. Czysty PyTorch (część 2/2)

```
37 def loss(model, x_int, x_bc, y_bc):
38     r = pde(model, x_int)
39     loss_pde = torch.mean(r**2)
40     loss_bc = torch.sum((model(x_bc) - y_bc) ** 2)
41     return loss_pde + loss_bc
42
43 def train_step(model, optimizer, x_int, x_bc, y_bc):
44     optimizer.zero_grad(set_to_none=True)
45     L = loss(model, x_int, x_bc, y_bc)
46     L.backward()
47     optimizer.step()
48     return L
49
50 x_bc = torch.tensor([[-1.0], [1.0]],
   ↪  dtype=torch.float32, device=device)
51 y_bc = torch.zeros_like(x_bc)
52
53 model = PINN().to(device)
54 optimizer = torch.optim.Adam(model.parameters(),
   ↪  lr=1e-3)
55
56 iterations = 10000
57 n_domain = 18
58 x_int = torch.linspace(-1.0, 1.0, steps=n_domain,
   ↪  dtype=torch.float32, device=device).view(-1, 1)
59
60 loss_history = []
61 for it in range(iterations):
62     L = train_step(model, optimizer, x_int, x_bc,
       ↪  y_bc)
63     loss_history.append(L.item())
64     if it % 1000 == 0:
65         print(f"Iter {it}: Loss = {L.item():.6e}")
```

Załącznik 8. Kod źródłowy rozwiązania zadania tarczy wspornikowej obciążonej równomiernie sieciami PINN

train.py

```python
"""python train.py [--config config.yaml]"""

import argparse
import logging
from pathlib import Path
from typing import Any

import yaml
import deepxde as dde

from src import (
    PlateParams,
    analytical_solution,
    build_bcs_mixed_soft,
    build_bcs_primal_soft,
    build_network,
    make_hard_bc_transform,
    make_pde_mixed,
    make_pde_primal,
)

logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s  %(levelname)-8s  %(message)s",
    datefmt="%H:%M:%S",
)
log = logging.getLogger(__name__)


def load_config(path: str | Path) -> dict[str, Any]:
    with open(path, "r", encoding="utf-8") as fh:
        cfg = yaml.safe_load(fh)
```

```
    log.info("Config loaded from '%s'", path)
    return cfg


def main(config_path: str | Path) -> None:
    cfg = load_config(config_path)

    outdir = Path(cfg["output"]["dir"])
    outdir.mkdir(parents=True, exist_ok=True)

    pp = PlateParams(cfg["physics"])
    pp.log_summary()

    mode    = cfg["formulation"]["mode"].lower()
    hard_bc = cfg["formulation"]["hard_bc"]

    if mode not in ("primal", "mixed"):
        raise ValueError(f"Unknown mode '{mode}'. Choose: primal | mixed")

    geom = dde.geometry.Rectangle([0.0, -0.5], [1.0, 0.5])
    pde  = make_pde_primal(pp) if mode == "primal" else make_pde_mixed(pp)

    if hard_bc:
        if mode == "primal":
            raise NotImplementedError("Hard BCs: mixed mode only.")
        bcs = []
        net = build_network(cfg["network"])
        net.apply_output_transform(make_hard_bc_transform(pp))
    else:
        bcs = (build_bcs_primal_soft(geom, pp) if mode == "primal"
               else build_bcs_mixed_soft(geom, pp))
        net = build_network(cfg["network"])

    smp  = cfg["sampling"]
    data = dde.data.PDE(
        geom, pde, bcs,
        num_domain   = smp["num_domain"],
        num_boundary = smp["num_boundary"],
        solution     = analytical_solution(pp, mode),
```

```
    num_test     = smp["num_test"],
)

model = dde.Model(data, net)

# Phase 1: Adam
adam_cfg = cfg["adam"]
model.compile("adam", lr=adam_cfg["lr"], metrics=["l2 relative error"])

callbacks = [
    dde.callbacks.EarlyStopping(
        monitor   = adam_cfg["early_stopping"]["monitor"],
        patience  = adam_cfg["early_stopping"]["patience"],
        min_delta = adam_cfg["early_stopping"]["min_delta"],
    ),
    dde.callbacks.ModelCheckpoint(
        filepath         = str(outdir / "model"),
        save_better_only = adam_cfg["checkpoint"]["save_better_only"],
        monitor          = adam_cfg["checkpoint"]["monitor"],
        period           = adam_cfg["checkpoint"]["period"],
    ),
]

log.info("=== Adam (%d iter) ===", adam_cfg["iterations"])
losshistory, train_state = model.train(
    iterations    = adam_cfg["iterations"],
    display_every = adam_cfg["display_every"],
    callbacks     = callbacks,
)

# Phase 2: L-BFGS-B
lbfgs_cfg = cfg.get("lbfgs", {})
if lbfgs_cfg.get("enabled", False):
    log.info("=== L-BFGS-B ===")
    model.compile("L-BFGS", metrics=["l2 relative error"])
    losshistory, train_state = model.train(
        display_every=lbfgs_cfg.get("display_every", 200),
    )
```

```
    dde.saveplot(
        losshistory, train_state,
        issave=True, isplot=False,
        output_dir=str(outdir),
    )
    log.info("Done. Results in '%s'.", outdir)


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--config", default="config.yaml")
    args = parser.parse_args()
    main(args.config)
```

src/params.py

```
import logging

import deepxde as dde

J, H = dde.grad.jacobian, dde.grad.hessian

log = logging.getLogger(__name__)

def make_ops(l: float, c: float):
    def Dxi    (f, xi_eta, i=0): return (1.0 / l)        * J(f, xi_eta, i=i, j=0)
    def Deta   (f, xi_eta, i=0): return (1.0 / (2 * c))  * J(f, xi_eta, i=i, j=1)
    def Dxixi  (f, xi_eta, i):   return (1.0 / l**2)     * H(f, xi_eta, component=i,
    ↪  i=0, j=0)
    def Detaeta(f, xi_eta, i):   return (1.0 / (2*c)**2) * H(f, xi_eta, component=i,
    ↪  i=1, j=1)
    return Dxi, Deta, Dxixi, Detaeta


class PlateParams:
    def __init__(self, phys: dict[str, float]) -> None:
        self.p  = phys["p"]       # line load           [kN/m]
        self.b  = phys["b"]       # plate thickness     [m]
        self.E  = phys["E"]       # Young's modulus     [kN/m²]
```

```
        self.nu = phys["nu"]      # Poisson's ratio     [-]
        self.l  = phys["l"]       # plate length        [m]
        self.c  = phys["c"]       # half-height         [m]
        self.k  = phys["k"]       # scaling exponent    [-]

        self.q      = self.p / self.b                    # surface load        [kN/m²]
        self.lam    = (2.0 * self.c) / self.l            # slenderness ratio   [-]
        self.sigma0 = self.q * (self.lam ** self.k)      # stress scale        [kN/m²]
        self.u0     = self.sigma0 * self.l / self.E      # displacement scale  [m]

        self.Dxi, self.Deta, self.Dxixi, self.Detaeta = make_ops(self.l, self.c)

    def log_summary(self) -> None:
        log.info(
            "Plate params:  q = %.2f kN/m²,  lam = %.4f,  "
            "sigma0 = %.4e kN/m²,  u0 = %.4e m",
            self.q, self.lam, self.sigma0, self.u0,
        )
```

src/solution.py

```
import numpy as np

from .params import PlateParams


def analytical_solution(pp: PlateParams, mode: str):
    def func(xi_eta: np.ndarray) -> np.ndarray:
        xi  = xi_eta[:, 0:1]
        eta = xi_eta[:, 1:2]
        x = pp.l       * xi
        y = 2.0 * pp.c * eta

        q, E, nu, l, c = pp.q, pp.E, pp.nu, pp.l, pp.c

        u_xi = (q / (20 * E * c**3)) * (
            5 * (x**3 - l**3) * y
            - 5 * (2 + nu) * x * y**3
            + 3 * (2 + 5 * nu) * c**2 * (x - l) * y
```

```
            + 10 * nu * c**3 * (x - l)
        )
        u_eta = (q / (80 * E * c**3)) * (
            5 * (1 + 2 * nu) * y**4
            - 30 * nu * x**2 * y**2
            - 6 * (5 + 2 * nu) * c**2 * y**2
            - 40 * c**3 * y
            + 30 * nu * c**2 * (x**2 - 3 * l**2)
            + 24 * c**2 * (2 * x**2 - 3 * l**2)
            - 5 * (x**4 + 3 * l**4)
            + 20 * l**3 * x
            + 12 * (2 + 5 * nu) * c**2 * l * x
        )

        if mode == "primal":
            return np.hstack((u_xi / pp.u0, u_eta / pp.u0))

        S_xixi   = (q / (20 * c**3)) * y * (15 * x**2 - 10 * y**2 + 6 * c**2)
        S_etaeta = (q / (4  * c**3)) * (y**3 - 3 * c**2 * y - 2 * c**3)
        S_xieta  = (3 * q / (4 * c**3)) * x * (c**2 - y**2)

        return np.hstack((
            u_xi     / pp.u0,
            u_eta    / pp.u0,
            S_xixi   / pp.sigma0,
            S_etaeta / pp.sigma0,
            S_xieta  / pp.sigma0,
        ))

    return func
```

src/pde.py

```
from .params import PlateParams
from src.params import PlateParams

def strains_and_stresses(f, xi_eta, pp: PlateParams):
    E_xixi   = pp.u0 * pp.Dxi (f, xi_eta, i=0)
    E_etaeta = pp.u0 * pp.Deta(f, xi_eta, i=1)
```

```
    E_xieta  = 0.5 * pp.u0 * (pp.Deta(f, xi_eta, i=0) + pp.Dxi(f, xi_eta, i=1))

    coeff = pp.E / (1.0 - pp.nu**2)
    S_xixi   = coeff * (E_xixi   + pp.nu * E_etaeta)
    S_etaeta = coeff * (E_etaeta + pp.nu * E_xixi)
    S_xieta  = (pp.E / (1.0 + pp.nu)) * E_xieta

    return S_xixi, S_etaeta, S_xieta

def make_pde_primal(pp: PlateParams):
    Dxi, Deta = pp.Dxi, pp.Deta

    def pde(x, f):
        S_xixi, S_etaeta, S_xieta = strains_and_stresses(f, x, pp)

        # Equilibrium:  div(sigma) = 0
        r_eqxi  = (Dxi(S_xixi,  x) + Deta(S_xieta,  x)) / pp.sigma0
        r_eqeta = (Dxi(S_xieta, x) + Deta(S_etaeta, x)) / pp.sigma0

        # Compatibility:  Laplacian(sigma_xx + sigma_yy) = 0
        r_comp = (
            Dxi (Dxi (S_xixi,   x), x) + Deta(Deta(S_xixi,   x), x)
            + Dxi(Dxi (S_etaeta, x), x) + Deta(Deta(S_etaeta, x), x)
        ) / pp.sigma0

        return [r_eqxi, r_eqeta, r_comp]

    return pde

def make_pde_mixed(pp: PlateParams):
    Dxi, Deta, Dxixi, Detaeta = pp.Dxi, pp.Deta, pp.Dxixi, pp.Detaeta

    def pde(x, f):
        S_xixi_phys, S_etaeta_phys, S_xieta_phys = strains_and_stresses(f, x, pp)

        # Constitutive:  sigma_Hooke / sigma0  -  sigma_net  = 0
        r_sxixi   = S_xixi_phys   / pp.sigma0 - f[:, 2:3]
        r_setaeta = S_etaeta_phys / pp.sigma0 - f[:, 3:4]
        r_sxieta  = S_xieta_phys  / pp.sigma0 - f[:, 4:5]
```

```
        # Equilibrium (on network stress outputs)
        r_eqxi  = Dxi(f, x, i=2) + Deta(f, x, i=4)
        r_eqeta = Dxi(f, x, i=4) + Deta(f, x, i=3)

        # Compatibility:  Laplacian(S_xixi + S_etaeta) = 0
        r_comp = Dxixi(f, x, 2) + Detaeta(f, x, 2) + Dxixi(f, x, 3) + Detaeta(f, x, 3)

        return [r_eqxi, r_eqeta, r_comp, r_sxixi, r_setaeta, r_sxieta]

    return pde
```

src/bcs.py

```
import numpy as np
import torch

import deepxde as dde

from src.params import PlateParams
from src.pde    import strains_and_stresses

J = dde.grad.jacobian


def _boundary_selectors():
    def left  (xi_eta, on_b): return on_b and dde.utils.isclose(xi_eta[0], 0.0)
    def right (xi_eta, on_b): return on_b and dde.utils.isclose(xi_eta[0], 1.0)
    def top   (xi_eta, on_b): return on_b and dde.utils.isclose(xi_eta[1], 0.5)
    def bottom(xi_eta, on_b): return on_b and dde.utils.isclose(xi_eta[1], -0.5)
    return left, right, top, bottom

def _point_bcs_displacement():
    """u(xi=1, eta=0) = 0,  du_xi/d_eta(xi=1, eta=0) = 0."""
    pt      = np.array([[1.0, 0.0]], dtype=np.float32)
    zero_pt = np.array([[0.0]],      dtype=np.float32)

    bc_uxi  = dde.icbc.PointSetBC(pt, zero_pt, component=0)
    bc_ueta = dde.icbc.PointSetBC(pt, zero_pt, component=1)
```

```
    def _no_rotation(xi_eta, f, _):
        return J(f, xi_eta, i=0, j=1)
    bc_rot = dde.icbc.PointSetOperatorBC(pt, zero_pt, _no_rotation)

    return [bc_uxi, bc_ueta, bc_rot]


def build_bcs_primal_soft(geom, pp: PlateParams) -> list:
    _left, _right, _top, _bottom = _boundary_selectors()

    def _S_xixi  (xi_eta, f, _): return strains_and_stresses(f, xi_eta, pp)[0] /
    ↪  pp.sigma0
    def _S_etaeta(xi_eta, f, _): return strains_and_stresses(f, xi_eta, pp)[1] /
    ↪  pp.sigma0
    def _S_xieta (xi_eta, f, _): return strains_and_stresses(f, xi_eta, pp)[2] /
    ↪  pp.sigma0

    bc_sxixi_left  = dde.icbc.OperatorBC(geom, _S_xixi,  _left)
    bc_sxieta_left = dde.icbc.OperatorBC(geom, _S_xieta, _left)

    bc_setaeta_bot = dde.icbc.OperatorBC(geom, _S_etaeta, _bottom)
    bc_sxieta_bot  = dde.icbc.OperatorBC(geom, _S_xieta,  _bottom)

    bc_setaeta_top = dde.icbc.OperatorBC(
        geom,
        lambda xi_eta, f, _: _S_etaeta(xi_eta, f, _) + pp.q / pp.sigma0,
        _top,
    )
    bc_sxieta_top = dde.icbc.OperatorBC(geom, _S_xieta, _top)

    return [
        *_point_bcs_displacement(),
        bc_sxixi_left,  bc_sxieta_left,
        bc_setaeta_bot, bc_sxieta_bot,
        bc_setaeta_top, bc_sxieta_top,
    ]

def build_bcs_mixed_soft(geom, pp: PlateParams) -> list:
```

```
    _left, _right, _top, _bottom = _boundary_selectors()
    q_hat = pp.q / pp.sigma0

    bc_sxixi_left  = dde.icbc.DirichletBC(geom, lambda x: 0.0,           _left,
    ↪  component=2)
    bc_sxieta_left = dde.icbc.DirichletBC(geom, lambda x: 0.0,           _left,
    ↪  component=4)
    bc_setaeta_bot = dde.icbc.DirichletBC(geom, lambda x: 0.0,           _bottom,
    ↪  component=3)
    bc_sxieta_bot  = dde.icbc.DirichletBC(geom, lambda x: 0.0,           _bottom,
    ↪  component=4)
    bc_setaeta_top = dde.icbc.DirichletBC(geom, lambda x: float(-q_hat), _top,
    ↪  component=3)
    bc_sxieta_top  = dde.icbc.DirichletBC(geom, lambda x: 0.0,           _top,
    ↪  component=4)

    return [
        *_point_bcs_displacement(),
        bc_sxixi_left,  bc_sxieta_left,
        bc_setaeta_bot, bc_sxieta_bot,
        bc_setaeta_top, bc_sxieta_top,
    ]


def make_hard_bc_transform(pp: PlateParams):
    """Mixed only. Enforces:
    u=0 at xi=1,  S_xixi=0 at xi=0,
    S_etaeta=0/-q_hat at eta=-0.5/+0.5,  S_xieta=0 at xi=0 and eta=+/-0.5.
    """
    q_hat = pp.q / pp.sigma0

    def transform(xi_eta, f):
        xi  = xi_eta[:, 0:1]
        eta = xi_eta[:, 1:2]
        U_xi     = f[:, 0:1] * (1.0 - xi)
        U_eta    = f[:, 1:2] * (1.0 - xi)
        S_xixi   = f[:, 2:3] * xi
        S_etaeta = -q_hat * (eta + 0.5) + f[:, 3:4] * (eta + 0.5) * (eta - 0.5)
        S_xieta  = f[:, 4:5] * xi * (eta + 0.5) * (eta - 0.5)
```

```
        return torch.cat([U_xi, U_eta, S_xixi, S_etaeta, S_xieta], dim=1)

    return transform
```

src/network.py

```
from typing import Any

import deepxde as dde


def build_network(net_cfg: dict[str, Any]) -> dde.nn.NN:
    arch        = net_cfg["type"].upper()
    layers      = net_cfg["layers"]
    activation  = net_cfg["activation"]
    initializer = net_cfg["initializer"]

    if arch == "FNN":
        return dde.nn.FNN(layers, activation, initializer)

    if arch == "PFNN":
        if not isinstance(layers[0], (list, tuple)):
            raise ValueError(
                "For PFNN, 'layers' must be a list of per-output layer lists, "
                "e.g. [[2,64,64,1], [2,64,64,1], ...]"
            )
        return dde.nn.PFNN(layers, activation, initializer)

    raise ValueError(f"Unknown network type '{arch}'. Choose 'FNN' or 'PFNN'.")
```

Załącznik 9. Kod źródłowy zadania identyfikacji parametrów materiałowych relacji konstytutywnych hipersprężystości na podstawie rozciągania elastomerowej płytki z otworem

config.yaml

```
output:
  dir: "output/T1W2-speckle-200-other-init-params"

physics:
  W: 50.0
  H: 40.0
  R: 10.0
  T: 2.0
  stretch: 3.0
  noise: 0.01
  num_speckle: 200
  holes_xy: [[0.0, 0.0]]
  force_band_x: [0.6, 0.8]

data:
  law_name: "yeoh"
  folder: "pinn/data/yeoh2"
  num_points_force: 400
  num_points_upper_lower: 100
  num_points_hole: 40

# Forward
# constitutive:
#   mode: "fixed"
#   law: "yeoh"
#   params: { C1: 0.5, C2: 0.03, C3: 0.005 }

# # Inverse
constitutive:
  mode: "variable"
```

```
  law: "yeoh"
  params: { C1: 2.0, C2: 0.5, C3: 0.3 }
  target_params: { C1: 0.5, C2: 0.03, C3: 0.005 }

# # CANN
# constitutive:
#   mode: "module"
#   law: "icnn"
#   params: { hidden: 32, n_layers: 3 }
#   checkpoint: null
#   freeze: false

network:
  type: "FNN"
  layers: [3, 32, 32, 32, 32, 3]
  activation: "tanh"
  initializer: "Glorot uniform"

sampling:
  num_domain: 30000
  anchors: 2000
  num_boundary: 0
  num_initial: 0
  num_test: 2000

loss_weights:
  incom: 10
  bof1: 1
  bof2: 1
  bc_upper_lower: 10
  bc_speckle_x: 2000
  bc_speckle_y: 400
  bc_right_force: 100
  bc_inner: 500

training:
  iterations: 30000
  lr: 0.001
  display_every: 1000
```

```
  checkpoint_every: 10000

lbfgs:
  enabled: true
  display_every: 200
```

train.py

```
"""python -m pinn.train [--config pinn/config/config.yaml]"""

import json
import argparse, logging, os, shutil
from pathlib import Path
import sys
sys.path.insert(0, str(Path(__file__).resolve().parent.parent))

import numpy as np
import pandas as pd
import yaml

os.environ["DDE_BACKEND"] = "pytorch"

import deepxde as dde
import torch

from pinn.src import (
    PlateHoleParams, make_pde, build_bcs, build_network,
    load_fem_data, classify_boundary_nodes,
    build_speckle_data, build_force_points,
    build_upper_lower_points, build_hole_points, build_hole_collocation_points
)
from pinn.src.integration_bc import IntegrationBC

from pinn.src.solution import build_solution_fn

logging.basicConfig(level=logging.INFO, format="%(asctime)s %(levelname)-8s
↪  %(message)s")
log = logging.getLogger(__name__)
```

```
def _build_constitutive(cfg):
    mode   = cfg["mode"]
    law    = cfg["law"]
    params = cfg.get("params", {})

    if mode == "fixed":
        from pinn.src.energy_adapters import FIXED_REGISTRY
        energy_fn, ext_vars, var_names = FIXED_REGISTRY[law](**params)
        log.info("Constitutive: FIXED %s, params=%s", law, params)

    elif mode == "variable":
        from pinn.src.energy_adapters import ADAPTER_REGISTRY
        energy_fn, ext_vars, var_names = ADAPTER_REGISTRY[law](**params)
        log.info("Constitutive: dde.Variable %s, %d trainable vars", law,
        ↪  len(ext_vars))

    elif mode == "module":
        from acle.src.constitutive import ENERGY_REGISTRY
        ckpt = cfg.get("checkpoint")
        if ckpt and Path(ckpt).exists():
            from acle.src.constitutive import load_constitutive_net
            energy_mod = load_constitutive_net(ckpt, arch=law, **params)
            energy_mod.train()
            log.info("Loaded CANN from %s", ckpt)
        else:
            energy_mod = ENERGY_REGISTRY[law](**params)
            log.info("Fresh CANN: %s", law)

        if cfg.get("freeze", False):
            for p in energy_mod.parameters():
                p.requires_grad_(False)
            log.info("CANN frozen.")

        energy_fn = energy_mod
        ext_vars = [p for p in energy_mod.parameters() if p.requires_grad]
        if hasattr(energy_mod, "named_material_params"):
            var_names = list(energy_mod.named_material_params().keys())
        else:
```

```
            var_names = [f"p{i}" for i in range(len(ext_vars))]

    else:
        raise ValueError(f"Unknown mode: {mode}")

    return energy_fn, ext_vars, var_names


def run_from_cfg(cfg, outdir=None, save_artifacts=True, use_variable_callback=True):
    if save_artifacts and outdir is None:
        raise ValueError("save_artifacts=True requires outdir to be set")
    outdir = Path(outdir) if outdir is not None else None

    pp = PlateHoleParams(**cfg["physics"])
    pp.log_summary()

    energy_fn, ext_vars, var_names = _build_constitutive(cfg["constitutive"])

    dcfg = cfg["data"]
    folder = dcfg["folder"]
    force_data = pd.read_csv(os.path.join(folder, "Force.csv"))
    time_list_raw = force_data["Time"].values
    force_list = force_data["Force"].values / float(pp.T * 2 * pp.H)
    num_time = len(time_list_raw)

    dic_node, _, dic_disp = load_fem_data(folder, num_time, pp)
    bc_up, bc_ri, bc_lo, bc_le, bc_in, bc_arc = classify_boundary_nodes(dic_node, pp)
    time_list = np.linspace(0, 1, num_time)

    spk_pts, spk_vx, spk_vy = build_speckle_data(
        dic_disp, bc_in, time_list, pp)
    force_pts, force_vals = build_force_points(
        time_list_raw, force_list, dcfg["num_points_force"], pp)
    ul_pts, ul_vals = build_upper_lower_points(
        time_list_raw, dcfg["num_points_upper_lower"], pp)
    hole_pts, hole_vals = build_hole_points(
        time_list, dcfg["num_points_hole"], pp)

    net = build_network(cfg["network"], pp)
```

```
# geometry
outer = dde.geometry.geometry_2d.Rectangle([-pp.w, -pp.h], [pp.w, pp.h])
spatial = outer
for h_xy in pp.holes:
    spatial = spatial - dde.geometry.geometry_2d.Disk(h_xy, pp.r)
geom = dde.geometry.GeometryXTime(spatial, dde.geometry.TimeDomain(0.0, 1.0))

pde = make_pde(energy_fn)
bcs = build_bcs(
    energy_fn, pp,
    points_upper_lower=ul_pts, value_upper_lower=ul_vals,
    points_force=force_pts, value_force=force_vals,
    num_time=num_time, num_points_force=dcfg["num_points_force"],
    inner_speckle_list=spk_pts,
    inner_speckle_list_value_x=spk_vx, inner_speckle_list_value_y=spk_vy,
    points_hole=hole_pts, value_hole=hole_vals,
    IntegrationBC=IntegrationBC,
)

smp = cfg["sampling"]
sol_fn = build_solution_fn(dic_node, dic_disp, time_list, pp)

anchors_n = smp.get("anchors")
anchors = (build_hole_collocation_points(time_list, num_points_hole=anchors_n,
↪  pp=pp)
           if anchors_n else None)

data = dde.data.TimePDE(geom, pde, bcs,
                        num_domain=smp["num_domain"],
                        num_boundary=smp["num_boundary"],
                        num_initial=smp["num_initial"],
                        num_test=smp["num_test"],
                        solution=sol_fn,
                        anchors=anchors)
model = dde.Model(data, net)

# loss weights
w = cfg["loss_weights"]
```

```
weights = [w["incom"], w["bof1"], w["bof2"], w["bc_upper_lower"],
           w["bc_speckle_x"], w["bc_speckle_y"], w["bc_right_force"]]
if pp.all_discs is not None and len(pp.all_discs) > 0:
    weights.append(w["bc_inner"])

def l2_rel_disp(y_true, y_pred):
    diff = y_true[:, :2] - y_pred[:, :2]
    return np.linalg.norm(diff) / np.linalg.norm(y_true[:, :2])

# compile
tcfg = cfg["training"]
model.compile("adam", lr=tcfg["lr"],
              external_trainable_variables=ext_vars,
              loss_weights=weights,
              metrics=[l2_rel_disp])

callbacks = []
if ext_vars and use_variable_callback and save_artifacts:
    callbacks.append(dde.callbacks.VariableValue(
        ext_vars, period=100,
        filename=str(outdir / "variable_history"), precision=9))

# Phase 1: Adam
log.info("=== Adam (%d iter) ===", tcfg["iterations"])
model.train(iterations=tcfg["iterations"],
            display_every=tcfg["display_every"],
            callbacks=callbacks)

# Phase 2: L-BFGS
lbfgs_cfg = cfg.get("lbfgs", {})
if lbfgs_cfg.get("enabled", False):
    log.info("=== L-BFGS-B ===")
    model.compile("L-BFGS",
                  external_trainable_variables=ext_vars,
                  loss_weights=weights,
                  metrics=[l2_rel_disp])
    model.train(display_every=lbfgs_cfg.get("display_every", 200),
                callbacks=callbacks)
```

```
# collect final parameters
mode = cfg["constitutive"]["mode"]
final_params = {}
if mode == "variable" and ext_vars:
    final_params = {name: float(v.item())
                    for name, v in zip(var_names, ext_vars)}
elif mode == "module" and hasattr(energy_fn, "named_material_params"):
    final_params = {k: float(v.item())
                    for k, v in energy_fn.named_material_params().items()}
elif mode == "fixed":
    final_params = dict(cfg["constitutive"]["params"])

final_metric = (float(model.losshistory.metrics_test[-1][0])
                if model.losshistory.metrics_test else float("inf"))

if save_artifacts:
    if mode == "module" and hasattr(energy_fn, "named_material_params"):
        log.info("Final constitutive parameters:")
        for k, v in energy_fn.named_material_params().items():
            log.info("  %s = %.9f", k, v.item())
    elif mode == "variable" and ext_vars:
        log.info("Final identified parameters:")
        for name, v in zip(var_names, ext_vars):
            log.info("  %s = %.9f", name, v.item())
    elif mode == "fixed":
        log.info("Fixed parameters (not trained): %s",
        ↪  cfg["constitutive"]["params"])

    model.save(str(outdir / "model"))
    if mode == "module":
        torch.save(energy_fn.state_dict(), str(outdir / "const_net_final.pt"))

    if final_params:
        with open(outdir / "final_params.json", "w") as f:
            json.dump(final_params, f, indent=2)
        log.info("Saved final_params.json")

    np.save(str(outdir / "steps.npy"), model.losshistory.steps)
    np.save(str(outdir / "loss_train.npy"), model.losshistory.loss_train)
```

```
        np.save(str(outdir / "loss_test.npy"), model.losshistory.loss_test)
        np.save(str(outdir / "loss_weights.npy"), np.array(weights, dtype=float))

        log.info("Done. Results in '%s'.", outdir)

    return {
        "final_params": final_params,
        "final_metric": final_metric,
        "var_names": var_names,
        "model": model,
    }


def main(config_path):
    with open(config_path) as f:
        cfg = yaml.safe_load(f)

    outdir = Path(cfg["output"]["dir"])
    outdir.mkdir(parents=True, exist_ok=True)
    shutil.copy2(config_path, outdir / Path(config_path).name)

    run_from_cfg(cfg, outdir=outdir, save_artifacts=True)


if __name__ == "__main__":
    p = argparse.ArgumentParser()
    p.add_argument("--config", default="pinn/config/config-T2.yaml")
    main(p.parse_args().config)
```

optuna_tune.py

```
import argparse
import copy
import logging
import sys
import traceback
from pathlib import Path

sys.path.insert(0, str(Path(__file__).resolve().parent.parent))
```

```
import yaml
import optuna

from pinn.train import run_from_cfg

logging.basicConfig(level=logging.INFO,
                    format="%(asctime)s %(levelname)-8s %(message)s")
log = logging.getLogger("optuna_tune")
optuna.logging.set_verbosity(optuna.logging.WARNING)


# Log-uniform search space for each loss weight.
WEIGHT_SPACE = {
    "incom":          (1.0,    1000.0),
    "bof1":           (0.1,     100.0),
    "bof2":           (0.1,     100.0),
    "bc_upper_lower": (1.0,    1000.0),
    "bc_speckle_x":   (1.0,    5000.0),
    "bc_speckle_y":   (1.0,    2000.0),
    "bc_right_force": (1.0,    1000.0),
    "bc_inner":       (0.1,    1000.0),
}

IDX_INCOM = 0
IDX_BC_INNER = 7


def suggest_weights(trial):
    return {name: trial.suggest_float(name, lo, hi, log=True)
            for name, (lo, hi) in WEIGHT_SPACE.items()}


def objective(trial, base_cfg):
    weights = suggest_weights(trial)
    log.info("Trial %d | weights: %s", trial.number,
             {k: f"{v:.3g}" for k, v in weights.items()})

    cfg = copy.deepcopy(base_cfg)
```

```
cfg["loss_weights"] = weights

try:
    result = run_from_cfg(cfg, save_artifacts=False,
                          use_variable_callback=False)
except Exception as e:
    log.warning("Trial %d crashed: %s", trial.number, e)
    log.debug(traceback.format_exc())
    return float("inf")

model = result["model"]
loss_history = model.losshistory

l2 = (float(loss_history.metrics_test[-1][0])
      if loss_history.metrics_test else float("inf"))

last_weighted = loss_history.loss_train[-1] if loss_history.loss_train else None
if last_weighted is None:
    log.warning("Trial %d: empty loss history", trial.number)
    return float("inf")

weight_list = [weights["incom"], weights["bof1"], weights["bof2"],
               weights["bc_upper_lower"], weights["bc_speckle_x"],
               weights["bc_speckle_y"], weights["bc_right_force"],
               weights["bc_inner"]]
raw_losses = [lw / w if w > 0 else lw
              for lw, w in zip(last_weighted, weight_list)]
incom_raw = raw_losses[IDX_INCOM]
bc_inner_raw = raw_losses[IDX_BC_INNER]

score = l2 + 0.1 * incom_raw + 0.1 * bc_inner_raw

if not (score == score) or score == float("inf"):
    log.warning("Trial %d: non-finite score (l2=%.4g, incom=%.4g, bc_inner=%.4g)",
                trial.number, l2, incom_raw, bc_inner_raw)
    return float("inf")

log.info("Trial %d | l2=%.4f incom=%.4f bc_inner=%.4f | score=%.4f",
         trial.number, l2, incom_raw, bc_inner_raw, score)
```

```
    return score


def summarize(study):
    log.info("=== Study summary ===")
    done = [t for t in study.trials
            if t.state == optuna.trial.TrialState.COMPLETE]
    log.info("Trials: %d total, %d completed", len(study.trials), len(done))
    try:
        best = study.best_trial
        log.info("Best trial #%d | score=%.4f", best.number, best.value)
        for k, v in best.params.items():
            log.info("  %s: %.4f", k, v)
    except ValueError:
        log.info("No completed trials.")


def save_best_weights(study, output_path):
    try:
        best = study.best_trial
    except ValueError:
        log.warning("No completed trials - not writing best_weights.yaml")
        return
    payload = {"loss_weights": {k: round(v, 6) for k, v in best.params.items()}}
    with open(output_path, "w") as f:
        yaml.safe_dump(payload, f, sort_keys=False)
    log.info("Wrote %s", output_path)


def main():
    p = argparse.ArgumentParser()
    p.add_argument("--config", default="pinn/config/config-T2.yaml")
    p.add_argument("--trials", type=int, default=50,
                   help="0 = just summarize existing study.")
    p.add_argument("--storage", default=None,
                   help="SQLite path. Default: <output.dir>/optuna/study.db")
    p.add_argument("--study-name", default="plate-notches")
    p.add_argument("--best-weights-out", default=None,
                   help="Default: <output.dir>/optuna/best_weights.yaml")
```

```
p.add_argument("--seed", type=int, default=42)
args = p.parse_args()

with open(args.config) as f:
    cfg = yaml.safe_load(f)

if cfg["constitutive"]["mode"] != "fixed":
    raise SystemExit("This tuning script targets constitutive.mode='fixed'.")

# Optuna artifacts go under <output.dir>/optuna/ by default
optuna_dir = Path(cfg["output"]["dir"]) / "optuna"
optuna_dir.mkdir(parents=True, exist_ok=True)

storage_path = args.storage or str(optuna_dir / "study.db")
best_weights_path = args.best_weights_out or str(optuna_dir / "best_weights.yaml")

Path(storage_path).parent.mkdir(parents=True, exist_ok=True)
storage = f"sqlite:///{storage_path}"

study = optuna.create_study(
    direction="minimize",
    study_name=args.study_name,
    storage=storage,
    load_if_exists=True,
    sampler=optuna.samplers.TPESampler(seed=args.seed),
)
log.info("Study ready: storage=%s, trials so far=%d",
         storage, len(study.trials))

if args.trials > 0:
    log.info("Running %d trials", args.trials)
    study.optimize(
        lambda t: objective(t, cfg),
        n_trials=args.trials,
        show_progress_bar=False,
    )

summarize(study)
save_best_weights(study, best_weights_path)
```

```
if __name__ == "__main__":
    main()
```

src/bcs.py

```
import torch
import torch.nn as nn
from deepxde.icbc.boundary_conditions import PointSetOperatorBC

from pinn.src.pde import compute_stress
from pinn.src.params import PlateHoleParams


def _make_bc_upper_lower(const_net: nn.Module):
    def bc_func(x, y, X):
        _, P12, _, P22, *_ = compute_stress(y, x, const_net)
        return torch.sqrt(P12**2 + P22**2)
    return bc_func


def _make_bc_inner(const_net: nn.Module, pp: PlateHoleParams):
    discs = torch.tensor(pp.all_discs, dtype=torch.float32)
    r = pp.r

    def bc_func(x, y, X):
        P11, P12, P21, P22, *_ = compute_stress(y, x, const_net)
        pos = x[:, :2]
        d = torch.norm(pos.unsqueeze(1) - discs.unsqueeze(0), dim=2)
        nearest = torch.argmin(d, dim=1)
        centers = discs[nearest]
        n = (pos - centers) / r
        n1, n2 = n[:, 0:1], n[:, 1:2]
        tn1 = n1 * P11 + n2 * P12
        tn2 = n1 * P21 + n2 * P22
        return torch.sqrt(tn1**2 + tn2**2)
    return bc_func
```

```
def _make_bc_force(const_net: nn.Module):
    def func(x, y, X):
        P11, *_ = compute_stress(y, x, const_net)
        return P11
    return func


def _bc_speckle_x(x, y, X):
    return y[:, 0:1]


def _bc_speckle_y(x, y, X):
    return y[:, 1:2]


def build_bcs(const_net, pp, *, points_upper_lower, value_upper_lower,
              points_force, value_force, num_time, num_points_force,
              inner_speckle_list, inner_speckle_list_value_x,
              inner_speckle_list_value_y, points_hole, value_hole,
              IntegrationBC):
    bcs = [
        PointSetOperatorBC(points_upper_lower, value_upper_lower,
                           _make_bc_upper_lower(const_net)),
        PointSetOperatorBC(inner_speckle_list, inner_speckle_list_value_x,
                           _bc_speckle_x),
        PointSetOperatorBC(inner_speckle_list, inner_speckle_list_value_y,
                           _bc_speckle_y),
        IntegrationBC(points_force, _make_bc_force(const_net),
                      value_force, num_time, num_points_force),
    ]

    if pp.all_discs is not None and len(pp.all_discs) > 0 and points_hole is not None
    ↪  and len(points_hole) > 0:
        bcs.append(
            PointSetOperatorBC(points_hole, value_hole,
                               _make_bc_inner(const_net, pp))
        )
```

```
    return bcs
```

src/data.py

```
import os, random
import numpy as np
import pandas as pd

from pinn.src.params import PlateHoleParams

def _inside_plate(x, y, pp, tol=1e-9):
    return (-pp.w - tol <= x <= pp.w + tol) and (-pp.h - tol <= y <= pp.h + tol)

def _inside_any_disc(x, y, pp, tol=1e-9):
    discs = pp.all_discs
    if len(discs) == 0:
        return False
    d = np.linalg.norm(discs - np.array([x, y]), axis=1)
    return bool(np.any(d < pp.r - tol))


def _on_any_arc(x, y, pp, tol=1e-4):
    discs = pp.all_discs
    if len(discs) == 0:
        return False
    d = np.linalg.norm(discs - np.array([x, y]), axis=1)
    return bool(np.any(np.abs(d - pp.r) < tol))

def _arc_fraction_inside(disc, pp, n_probe=720):
    theta = np.linspace(0, 2 * np.pi, n_probe, endpoint=False)
    xc = disc[0] + pp.r * np.cos(theta)
    yc = disc[1] + pp.r * np.sin(theta)
    inside = np.array([_inside_plate(x, y, pp) for x, y in zip(xc, yc)])
    return inside.mean()


def load_fem_data(folder, num_time, pp):
    dic_node = np.load(os.path.join(folder, "dic_node.npy"),
```

```
                    allow_pickle=True, encoding="bytes").item()
    for k, (x, y) in dic_node.items():
        dic_node[k] = np.array([x / pp.S, y / pp.S], dtype=float)

    dic_connectivity = np.load(os.path.join(folder, "dic_connectivity.npy"),
                               allow_pickle=True, encoding="bytes").item()

    dic_displacement = []
    for i in range(num_time):
        disp = np.load(os.path.join(folder, f"{i}_dic_displacement.npy"),
                       allow_pickle=True, encoding="bytes").item()
        for k, v in disp.items():
            disp[k] = np.array([float(v[0]) / pp.S, float(v[1]) / pp.S], dtype=float)
        dic_displacement.append(disp)

    return dic_node, dic_connectivity, dic_displacement


def classify_boundary_nodes(dic_node, pp, edge_tol=1e-6, arc_tol=1e-4):
    bc_upper, bc_right, bc_lower, bc_left, bc_inner, bc_arc = [], [], [], [], [], []
    discs = pp.all_discs

    for index, (x, y) in dic_node.items():
        if len(discs) > 0:
            dist = np.linalg.norm(discs - np.array([x, y]), axis=1) - pp.r
            on_arc = bool(np.any(np.abs(dist) < arc_tol))
        else:
            on_arc = False

        if on_arc:
            bc_arc.append([index, x, y])
            continue

        if   np.isclose(x, -pp.w, atol=edge_tol): bc_left.append([index, x, y])
        elif np.isclose(x,  pp.w, atol=edge_tol): bc_right.append([index, x, y])
        elif np.isclose(y, -pp.h, atol=edge_tol): bc_lower.append([index, x, y])
        elif np.isclose(y,  pp.h, atol=edge_tol): bc_upper.append([index, x, y])
        else:
            bc_inner.append([index, x, y])
```

```
    def _sorted(lst, col, reverse=False):
        if not lst:
            return np.empty((0, 3))
        arr = np.array(lst)
        order = np.argsort(arr[:, col])
        if reverse:
            order = order[::-1]
        return arr[order]

    bc_upper = _sorted(bc_upper, 1)
    bc_right = _sorted(bc_right, 2, reverse=True)
    bc_lower = _sorted(bc_lower, 1, reverse=True)
    bc_left  = _sorted(bc_left,  2)
    bc_arc   = np.array(bc_arc) if bc_arc else np.empty((0, 3))

    if pp.num_speckle > len(bc_inner):
        raise ValueError(
            f"num_speckle={pp.num_speckle} exceeds number of interior nodes "
            f"({len(bc_inner)})."
        )
    bc_inner = np.array(bc_inner)
    bc_inner = bc_inner[np.random.choice(len(bc_inner), size=pp.num_speckle,
    ↪  replace=False)]

    return bc_upper, bc_right, bc_lower, bc_left, bc_inner, bc_arc



def build_speckle_data(dic_displacement, bc_inner, time_list, pp):
    num_time = len(time_list)
    x_all, y_all, c_all = [], [], []

    for i in range(num_time):
        xi, yi, ci = [], [], []
        for ele in bc_inner:
            idx = int(ele[0])
            ex = pp.noise * np.random.random() * (float(np.random.random() > 0.5) * 2
            ↪  - 1) + 1
```

```
            ey = pp.noise * np.random.random() * (float(np.random.random() > 0.5) * 2
            ↪  - 1) + 1
            xi.append(ele[1] + dic_displacement[i][idx][0] * ex)
            yi.append(ele[2] + dic_displacement[i][idx][1] * ey)
            ci.append([ele[1], ele[2], time_list[i]])
        x_all.append(xi); y_all.append(yi); c_all.append(ci)

    c_all = np.array(c_all)
    n = c_all.shape[0] * c_all.shape[1]
    return c_all.reshape(n, 3), np.array(x_all).reshape(n, 1),
    ↪  np.array(y_all).reshape(n, 1)


def build_force_points(time_list_raw, force_list, num_points_force, pp):
    lo, hi = pp.force_band_x
    xf = [(np.random.random() * (hi - lo) + lo) * pp.w * (1 if np.random.random() >
    ↪  0.5 else -1)
          for _ in range(num_points_force)]
    yf = [(np.random.random() * 2 - 1) * pp.h for _ in range(num_points_force)]
    pts, vals = [], []
    for i in range(len(time_list_raw)):
        for j in range(num_points_force):
            pts.append([xf[j], yf[j], time_list_raw[i]])
            vals.append([force_list[i]])
    return np.array(pts), np.array(vals)


def build_upper_lower_points(time_list_raw, num_points, pp, safety_margin=None):
    dx = 2.0 * pp.w / num_points
    if safety_margin is None:
        safety_margin = 2 * dx
    xs = np.linspace(-pp.w + dx / 2, pp.w - dx / 2, num_points)
    pts, vals = [], []
    for t in time_list_raw:
        for x in xs:
            for y in (-pp.h, pp.h):
                if _inside_any_disc(x, y, pp):
                    continue
                too_close = False
```

```
                for disc in pp.all_discs:
                    dist_to_center = np.linalg.norm(np.array([x, y]) - disc)
                    if abs(dist_to_center - pp.r) < safety_margin:
                        too_close = True
                        break
                if too_close:
                    continue
                pts.append([x, y, t]); vals.append([0.0])
    return np.array(pts), np.array(vals)



def build_hole_points(time_list, num_points_hole, pp, endpoint_skip=2):
    pts, vals = [], []
    for disc in pp.all_discs:
        cx, cy = float(disc[0]), float(disc[1])
        frac = _arc_fraction_inside(disc, pp)
        n = max(1, int(round(num_points_hole * frac)))
        theta = np.linspace(0, 2 * np.pi, int(round(n / frac)), endpoint=False)
        xc = cx + pp.r * np.cos(theta)
        yc = cy + pp.r * np.sin(theta)
        mask = np.array([_inside_plate(x, y, pp) for x, y in zip(xc, yc)])
        if endpoint_skip > 0 and mask.sum() > 2 * endpoint_skip:
            idxs = np.where(mask)[0]
            keep = idxs[endpoint_skip:-endpoint_skip] if endpoint_skip < len(idxs)//2
            ↪  else idxs
            new_mask = np.zeros_like(mask)
            new_mask[keep] = True
            mask = new_mask
        xc, yc = xc[mask], yc[mask]
        for t in time_list:
            for x, y in zip(xc, yc):
                pts.append([x, y, float(t)]); vals.append([0.0])
    return np.array(pts, dtype=np.float32), np.array(vals, dtype=np.float32)


def build_hole_collocation_points(time_list, num_points_hole, pp, r_max=0.3):
    pts = []
    for disc in pp.all_discs:
```

```
            cx, cy = float(disc[0]), float(disc[1])
            frac = _arc_fraction_inside(disc, pp)
            n_target = max(1, int(round(num_points_hole * frac)))
            for t in time_list:
                count = 0
                max_tries = 100 * n_target
                tries = 0
                while count < n_target and tries < max_tries:
                    r = pp.r + np.random.random() * r_max
                    theta = np.random.random() * 2 * np.pi
                    x = cx + r * np.cos(theta)
                    y = cy + r * np.sin(theta)
                    tries += 1
                    if _inside_plate(x, y, pp):
                        pts.append([x, y, float(t)])
                        count += 1
                if count < n_target:
                    raise RuntimeError(
                        f"Failed to draw {n_target} collocation points for "
                        f"disc at ({cx}, {cy}). Check r_max and geometry."
                    )
    return np.array(pts, dtype=np.float32)
```

src/energy_adapters.py

```
from __future__ import annotations

import torch
import deepxde as dde

def yeoh_fixed(C1, C2, C3):
    C1 = torch.tensor(C1, dtype=torch.float64)
    C2 = torch.tensor(C2, dtype=torch.float64)
    C3 = torch.tensor(C3, dtype=torch.float64)

    def energy_fn(I1, I2, J):
        I1min3 = I1 - 3.0
        W = C1*I1min3 + C2*I1min3**2 + C3*I1min3**3
```

```
        dW_dI1 = C1 + 2.0*C2*I1min3 + 3.0*C3*(I1min3**2)
        return {
            "W": W,
            "dW_dI1": dW_dI1,
            "dW_dI2": torch.zeros_like(I1),
        }

    return energy_fn, [], []




FIXED_REGISTRY = {
    "yeoh":          yeoh_fixed,
}


def yeoh_adapter(C1=1.0, C2=0.1, C3=0.01):
    C1 = dde.Variable(C1)
    C2 = dde.Variable(C2)
    C3 = dde.Variable(C3)

    def energy_fn(I1, I2, J):
        I1min3 = I1 - 3.0
        W = C1*I1min3 + C2*I1min3**2 + C3*I1min3**3
        dW_dI1 = C1 + 2.0*C2*I1min3 + 3.0*C3*(I1min3**2)
        return {"W": W, "dW_dI1": dW_dI1, "dW_dI2": torch.zeros_like(I1)}

    return energy_fn, [C1, C2, C3], ["C1", "C2", "C3"]


ADAPTER_REGISTRY = {
    "yeoh":          yeoh_adapter,
}
```

src/integration_bc.py

```
import numpy as np
import deepxde as dde
```

```python
from deepxde import backend as bkd, config
from deepxde.icbc.boundary_conditions import BC


class IntegrationBC(BC):

    def __init__(self, points, func, values, num_time, num_points):
        self.points = np.array(points, dtype=config.real(np))
        self.values = bkd.as_tensor(values, dtype=config.real(bkd.lib))
        self.func = func
        self.num_time = num_time
        self.num_points = num_points

    def collocation_points(self, X):
        return self.points

    def error(self, X, inputs, outputs, beg, end):
        sol_raw = self.func(inputs, outputs, X)[beg:end] - self.values

        total = 0.0
        for i in range(self.num_time):
            s = self.num_points * i
            e = self.num_points * (i + 1)
            block_mean = bkd.mean(bkd.reshape(sol_raw[s:e], (-1,)), dim=0)
            total = total + bkd.abs(block_mean)

        scalar_loss = total / float(self.num_time)

        zeros = outputs[beg:end, 0:1] * 0.0
        return zeros + scalar_loss
```

src/network.py

```python
from typing import Any
import deepxde as dde
import torch

from pinn.src.params import PlateHoleParams
```

```
def _make_output_transform(pp: PlateHoleParams):
    stretch = pp.stretch

    def transform(x, y):
        X1, X2, t = x[:, 0:1], x[:, 1:2], x[:, 2:3]
        ratio = (stretch - 1.0) * t + 1.0
        new_x1 = (X1 + 1.0) * (X1 - 1.0) * y[:, 0:1] + (X1 + 1.0) * ratio - 1.0
        new_x2 = (X1 + 1.0) * (X1 - 1.0) * y[:, 1:2] + X2
        return torch.cat([new_x1, new_x2, y[:, 2:3]], dim=1)

    return transform


def build_network(net_cfg: dict[str, Any], pp: PlateHoleParams) -> dde.nn.NN:
    arch = net_cfg["type"].upper()
    layers = net_cfg["layers"]
    activation = net_cfg["activation"]
    initializer = net_cfg["initializer"]

    if arch == "FNN":
        net = dde.nn.FNN(layers, activation, initializer)
    elif arch == "PFNN":
        n_in, hidden, n_out = layers[0], layers[1], layers[2]
        pfnn_layers = [n_in] + [[h] * n_out for h in hidden] + [n_out]
        net = dde.nn.PFNN(pfnn_layers, activation, initializer)
    else:
        raise ValueError(f"Unknown network type '{arch}'.")

    net.apply_output_transform(_make_output_transform(pp))
    return net
```

src/params.py

```
import logging
from dataclasses import dataclass, field
from typing import List
import numpy as np
```

```
log = logging.getLogger(__name__)


@dataclass
class PlateHoleParams:
    W: float = 50.0
    H: float = 40.0
    R: float = 10.0
    T: float = 2.0
    holes_xy: List[List[float]] = field(default_factory=lambda: [[0.0, 0.0]])
    stretch: float = 3.0
    noise: float = 0.01
    num_speckle: int = 80
    force_band_x: tuple = (0.6, 0.8)

    geom_tol: float = 1e-9

    S: float = field(init=False)
    w: float = field(init=False)
    h: float = field(init=False)
    r: float = field(init=False)
    holes: np.ndarray = field(init=False, repr=False)
    edge_notches: np.ndarray = field(init=False, repr=False)

    def __post_init__(self):
        self.S = float(max(self.W, self.H))
        self.w = self.W / self.S
        self.h = self.H / self.S
        self.r = self.R / self.S

        self.force_band_x = tuple(self.force_band_x)
        if len(self.force_band_x) != 2:
            raise ValueError(f"force_band_x must have 2 elements, got
            ↪  {len(self.force_band_x)}")
        lo, hi = self.force_band_x
        if not (0 <= lo < hi <= 1):
            raise ValueError(f"force_band_x must satisfy 0 <= lo < hi <= 1, got
            ↪  {self.force_band_x}")
```

```
        centers = (np.array(self.holes_xy, dtype=float) / self.S
                   if self.holes_xy else np.empty((0, 2)))

        holes, notches = [], []
        for c in centers:
            kind = self._classify(c)
            (holes if kind == "interior" else notches).append(c)

        self.holes = np.array(holes) if holes else np.empty((0, 2))
        self.edge_notches = np.array(notches) if notches else np.empty((0, 2))



def _edge_distances(self, cx: float, cy: float):
    return [
        ("left",   cx - (-self.w)),
        ("right",  self.w - cx),
        ("bottom", cy - (-self.h)),
        ("top",    self.h - cy),
    ]

def _classify(self, center: np.ndarray) -> str:
    cx, cy = float(center[0]), float(center[1])
    r, tol = self.r, self.geom_tol
    dists = self._edge_distances(cx, cy)

    if any(d < -tol for _, d in dists):
        raise ValueError(
            f"Disc center ({cx:.4f}, {cy:.4f}) lies outside the plate "
            f"[-{self.w}, {self.w}] x [-{self.h}, {self.h}]."
        )

    tangent = [name for name, d in dists if abs(d - r) < tol]
    if tangent:
        raise ValueError(
            f"Disc at ({cx:.4f}, {cy:.4f}) is tangent to edge(s) "
            f"{tangent}. Tangency is not allowed - use either an interior "
            f"hole or a notch that crosses an edge."
```

```
    )

crossed = []
for name, d in dists:
    if d >= r:
        continue
    half_chord = float(np.sqrt(max(r * r - d * d, 0.0)))
    if name in ("left", "right"):
        lo, hi = cy - half_chord, cy + half_chord
        bound_lo, bound_hi, axis = -self.h, self.h, "y"
    else:
        lo, hi = cx - half_chord, cx + half_chord
        bound_lo, bound_hi, axis = -self.w, self.w, "x"
    if lo < bound_lo - tol or hi > bound_hi + tol:
        raise ValueError(
            f"Disc at ({cx:.4f}, {cy:.4f}) extends past the ends of "
            f"edge '{name}' ({axis}=[{lo:.4f}, {hi:.4f}], bounds "
            f"[{bound_lo}, {bound_hi}]). A notch must cross exactly "
            f"one edge."
        )
    crossed.append(name)

if len(crossed) == 0:
    self._check_clear_of_force_band(cx, cy)
    return "interior"
if len(crossed) == 1:
    edge = crossed[0]
    if edge in ("left", "right"):
        raise ValueError(
            f"Notch at ({cx:.4f}, {cy:.4f}) is on the '{edge}' edge. "
            f"Left and right edges are reserved for the force boundary "
            f"condition - notches are only allowed on top/bottom."
        )
    self._check_clear_of_force_band(cx, cy)
    return "edge_notch"
raise ValueError(
    f"Disc at ({cx:.4f}, {cy:.4f}) crosses edges {crossed}. "
    f"Only single-edge notches are supported."
)
```

```
def _check_clear_of_force_band(self, cx: float, cy: float) -> None:
    x_force_min = float(self.force_band_x[0]) * self.w
    if abs(cx) + self.r > x_force_min + self.geom_tol:
        raise ValueError(
            f"Disc at ({cx:.4f}, {cy:.4f}) with r={self.r:.4f} reaches "
            f"|x|={abs(cx) + self.r:.4f}, which overlaps the force-averaging "
            f"band starting at |x|={x_force_min:.4f}. Move the disc closer "
            f"to the center or widen the force band "
            f"(current force_band_x={self.force_band_x})."
        )

@property
def all_discs(self) -> np.ndarray:
    if len(self.holes) == 0:
        return self.edge_notches.copy()
    if len(self.edge_notches) == 0:
        return self.holes.copy()
    return np.vstack([self.holes, self.edge_notches])

@property
def has_notches(self) -> bool:
    return len(self.edge_notches) > 0

@property
def has_interior_holes(self) -> bool:
    return len(self.holes) > 0

def log_summary(self):
    log.info("PlateHole: W=%g H=%g R=%g T=%g stretch=%.2f w=%.4f h=%.4f r=%.4f",
             self.W, self.H, self.R, self.T, self.stretch, self.w, self.h, self.r)
    for i, xy in enumerate(self.holes):
        log.info("  hole[%d]:   center=(%.4f, %.4f)", i, xy[0], xy[1])
    for i, xy in enumerate(self.edge_notches):
        log.info("  notch[%d]:  center=(%.4f, %.4f)", i, xy[0], xy[1])
```

src/pde.py

```
import torch
import deepxde as dde

J = dde.grad.jacobian


def compute_stress(y, x, energy_fn):
    F33 = y[:, 2:3]
    F11 = J(y, x, i=0, j=0)
    F12 = J(y, x, i=0, j=1)
    F21 = J(y, x, i=1, j=0)
    F22 = J(y, x, i=1, j=1)

    J_det = (F11 * F22 - F12 * F21) * F33

    I1 = F11**2 + F12**2 + F21**2 + F22**2 + F33**2

    B11 = F11**2 + F12**2
    B12 = F11*F21 + F12*F22
    B21 = B12
    B22 = F21**2 + F22**2
    B33 = F33**2
    trB2 = B11**2 + 2*B12**2 + B22**2 + B33**2
    I2 = 0.5 * (I1**2 - trB2)

    out     = energy_fn(I1, I2, J_det)
    beta1   =  2.0 * out["dW_dI1"]   # 2 dW/dI1
    beta_m1 = -2.0 * out["dW_dI2"]   # -2 dW/dI2  (znak konwencji Ogden)


    p = beta1 * B33 + beta_m1 / B33.clamp(min=1e-8)

    BF11 = B11*F11 + B12*F21
    BF12 = B11*F12 + B12*F22
    BF21 = B21*F11 + B22*F21
    BF22 = B21*F12 + B22*F22
```

```
    pF3 = p * F33

    P11 = -pF3 *  F22 + beta1 * F11 + beta_m1 * (I1*F11 - BF11)
    P12 =  pF3 *  F21 + beta1 * F12 + beta_m1 * (I1*F12 - BF12)
    P21 =  pF3 *  F12 + beta1 * F21 + beta_m1 * (I1*F21 - BF21)
    P22 = -pF3 *  F11 + beta1 * F22 + beta_m1 * (I1*F22 - BF22)

    return P11, P12, P21, P22, F33, J_det


def make_pde(energy_fn):
    """PDE residual: [incompressibility, equilibrium_x, equilibrium_y]."""
    def pde(x, y):
        P11, P12, P21, P22, _, J_det = compute_stress(y, x, energy_fn)

        incom = J_det - 1.0
        BoF1 = J(P11, x, i=0, j=0) + J(P12, x, i=0, j=1)
        BoF2 = J(P21, x, i=0, j=0) + J(P22, x, i=0, j=1)

        return [incom, BoF1, BoF2]
    return pde
```

src/solution.py

```
from __future__ import annotations

import numpy as np
from scipy.interpolate import LinearNDInterpolator, NearestNDInterpolator


def build_solution_fn(dic_node, dic_displacement, time_list, pp,
                      num_outputs: int = 2):
    time_arr = np.asarray(time_list, dtype=np.float64)
    node_ids = sorted(dic_node.keys())
    coords = np.array([dic_node[nid] for nid in node_ids], dtype=np.float64)

    ux_steps, uy_steps = [], []
    for step_disp in dic_displacement:
        ux_steps.append(np.array([step_disp[nid][0] for nid in node_ids],
```

```
                                dtype=np.float64))
        uy_steps.append(np.array([step_disp[nid][1] for nid in node_ids],
                                dtype=np.float64))
    ux_steps = np.stack(ux_steps, axis=0)   # (T, N_nodes)
    uy_steps = np.stack(uy_steps, axis=0)

    interps_ux, interps_uy = [], []
    for t_idx in range(len(time_arr)):
        lin_ux = LinearNDInterpolator(coords, ux_steps[t_idx])
        near_ux = NearestNDInterpolator(coords, ux_steps[t_idx])
        lin_uy = LinearNDInterpolator(coords, uy_steps[t_idx])
        near_uy = NearestNDInterpolator(coords, uy_steps[t_idx])
        interps_ux.append((lin_ux, near_ux))
        interps_uy.append((lin_uy, near_uy))

    def solution(X: np.ndarray) -> np.ndarray:
        N = X.shape[0]
        result = np.zeros((N, num_outputs), dtype=np.float64)

        mat_X = X[:, 0]
        mat_Y = X[:, 1]
        t = X[:, 2]

        idx_right = np.searchsorted(time_arr, t, side="right")
        idx_right = np.clip(idx_right, 0, len(time_arr) - 1)
        idx_left = np.clip(idx_right - 1, 0, len(time_arr) - 1)

        t_lo = time_arr[idx_left]
        t_hi = time_arr[idx_right]
        dt = t_hi - t_lo
        dt[dt == 0] = 1.0
        alpha = np.clip((t - t_lo) / dt, 0.0, 1.0)

        xy = X[:, :2]

        brackets = np.unique(
            np.column_stack([idx_left, idx_right]), axis=0)

        for il, ir in brackets:
```

```
            il, ir = int(il), int(ir)
            mask = (idx_left == il) & (idx_right == ir)
            pts = xy[mask]

            ux_lo = _interp_batch(interps_ux[il], pts)
            uy_lo = _interp_batch(interps_uy[il], pts)

            if il == ir:
                ux = ux_lo
                uy = uy_lo
            else:
                ux_hi = _interp_batch(interps_ux[ir], pts)
                uy_hi = _interp_batch(interps_uy[ir], pts)
                a = alpha[mask]
                ux = (1 - a) * ux_lo + a * ux_hi
                uy = (1 - a) * uy_lo + a * uy_hi

            result[mask, 0] = mat_X[mask] + ux
            result[mask, 1] = mat_Y[mask] + uy

        return result

    return solution


def _interp_batch(interp_pair, pts):
    lin, near = interp_pair
    vals = lin(pts)
    nan_mask = np.isnan(vals)
    if np.any(nan_mask):
        vals[nan_mask] = near(pts[nan_mask])
    return vals
```