Biblioteka w Julii do modelowania układu sercowo-naczyniowego w oparciu o sieci neuronowe oparte na fizyce - Prototyp

Wiktor Perczak, Michał Dydek

Czerwiec 2025

1 Wstęp

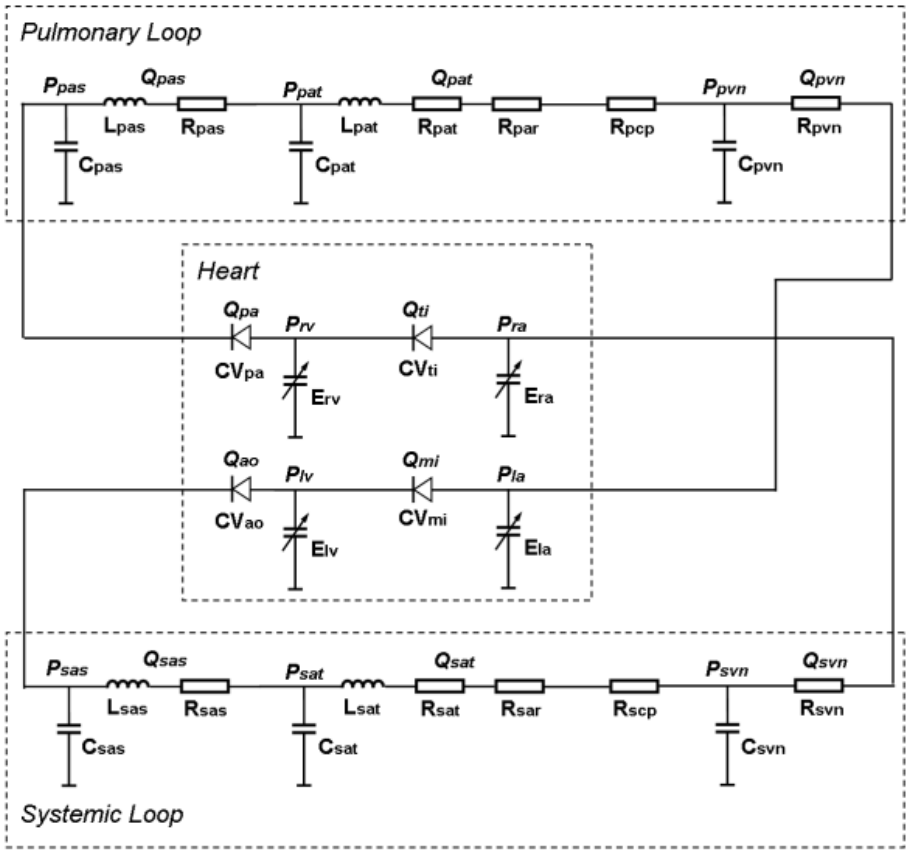
Dynamiczny rozwój technologii wraz z postępującym zapotrzebowaniem na jak najdokładniejsze dane medyczne sprawił, że personalizacja medycyny staje sie coraz bardziej fundamentalnym elementem profilaktyki i leczenia. Dodatek informatyki jako dziedziny nauki jest uważany za znaczące przyspieszenie i udoskonalenie tego procesu. Dotychczas stosowane narzędzia opierają się na podejściu bazującym na przetwarzaniu danych (uczenie maszynowe), lub wykorzystującym klasyczne modelowanie prawami fizycznymi (równania różniczkowe, mechanika płynów). Innowacyjnym podejściem, łączącym wcześniej wspomniane rozwiązania jest zastosowanie metod hybrydowych, takich jak Sieci Neurowe Oparte na Fizyce (PINN). Korzystają one zarówno z modeli fizycznych jak i sieci neuronowych, co pozwala im osiągnąć lepsze rezultaty niż stosowanie każdej z metod osobno.

2 Sformułowanie problemu

Jednym z problemów współczesnej medycyny jest modelowanie układu sercowo-naczyniowego, które jest niezwykle pomocne w profilaktyce i wczesnym wykrywaniu nieprawidłowości zdrowotnych. Istnieją bardzo dokładne odwzorowania tego systemu, lecz ich odpowiednie stosowanie, wymaga dużego nakładu pracy i zasobów. Rozwiązaniem tego problemu jest użycie mniej skomplikowanych modeli, których wadą jest znaczna utrata precyzji, nieakceptowalna w dziedzinie medycyny, gdzie na szali stoi ludzkie zdrowie. Użycie sieci neuronowych opartych na fizyce wraz z uproszczonym modelem, może sprawić, że otrzymamy rozwiązanie równie skuteczne jak pełne odwzorowanie układu sercowo-naczyniowego, biorącego pod uwagę wszystkie możliwe fizyczne zależności. Dlatego też w niniejszej pracy podjęto się stworzenia narzędzia ułatwiającego korzystanie z wyżej wymienionego podejścia.

3 Opis produktu

Cel pracy to biblioteka w języku Julia, umożliwiająca użytkownikowi w łatwy i szybki sposób wygenerować model układu sercowo-naczyniowego z wykorzystaniem PINN dla zadanych danych. Zaletą stosowania będzie prosty interfejs, bez konieczności indywidualnej implementacji systemu.



Rysunek 1: Czterokomorowy model układu sercowo-naczyniowego

Na ten moment nie istnieją narzędzia oferujące takie funkcjonalności, a mogłyby być one bardzo pomocne zarówno dla osób technicznych, przy dalszym rozwoju modeli związanych z układem sercowo-naczyniowym, jak i dla osób nietechnicznych. Przejrzysty i nieskomplikowany interfejs ma umożliwić proste przekazanie parametrów i otrzymanie wyniku.

4 Analiza użytych technologii

1. System kontroli wersji: GitHub. Na oficjalnej stronie języka sami twórcy zalecają używanie tej platformy, ponieważ późniejsze udostępnienie biblioteki szerszemu gronu jest bardzo proste, dzięki GitHub Actions i Julia Registrator.

2. Licencja: MIT License, prosta w "użyciu", darmowa, szeroko udostępniająca naszą pracę, zezwalająca na dowolne użytkowanie, rozpowszechnianie

3. Zarządzanie dependencjami: Julia Package Manager. W zasadzie dla języka Julia nie ma zbytnich alternatyw, jest to bardzo intuicyjny, wspierany przez autorów sposób na zarządzanie zależnościami.

4. Testowanie: na ten moment jeszcze ciężko przewidzieć jaka będzie finalna struktura biblioteki i co będziemy chcieli w niej zawrzeć, stąd na ten moment brak obecności testów. Natomiast jeśli zajdzie potrzeba dodania wygodna biblioteka "Test"również istnieje.

5. Udostępnianie biblioteki: Finalnie, żeby udostępnić bibliotekę potrzeba ją zarejestrować w "Julia General Registry". *Tutaj są opisane dokładne kroki*

5 Zastosowane biblioteki w projekcie

- BenchmarkTools - śledzenie wydajności kodu, czasu, zużycia zasobów systemowych
- CellMLToolkit - podstawowa biblioteka na której opieramy swoje działania, udostępnia mnóstwo modeli i solwerów dla równań różniczkowych
- ComponentArrays - umożliwia tworzenie i manipulowanie tablicami z nazwanymi osiami, co ułatwia pracę ze strukturalnymi danymi
- DelimitedFiles - biblioteka standardowa do odczytu i zapisu plików tekstowych z danymi rozdzielonymi, takich jak pliki CSV
- ForwardDiff - służy do automatycznego różniczkowania, umożliwiając obliczanie pochodnych funkcji względem parametrów wejściowych
- JuliaFormatter - biblioteka do formatowania plików Julia
- Lux - biblioteka do uczenia głębokiego w Julia, koncentrująca się na programowaniu różnicowym i dynamicznych sieciach neuronowych
- Measures - udostępnia narzędzia do określania wymiarów i rozmiarów z jednostkami, przydatne w układach graficznych i wykresach
- ModelingToolkit - biblioteka do obliczeń symbolicznych dla modelowania opartego na równaniach, umożliwiająca upraszczanie i transformację modeli
- Optim, Optimization, OptimizationOptimJL, OptimizationOptimisers - kilka paczek, których zadaniem jest rozwiązywanie problemów optymalizacyjnych
- OrdinaryDiffEq - biblioteka do rozwiązywania równań różniczkowych zwyczajnych (ODE) z kolekcją wysokowydajnych solverów
- Plots - biblioteka do tworzenia wykresów w Julia, oferująca uproszczony interfejs do tworzenia wizualizacji
- StableRNGs - dostarcza generatory liczb losowych o stabilnych strumieniach między wersjami Julia, zapewniając powtarzalność
- Statistics - biblioteka standardowa oferująca podstawowe funkcje statystyczne, takie jak średnia, mediana, odchylenie standardowe itd
- Zygote - biblioteka do automatycznego różniczkowania w Julia, zdolna do różniczkowania natywnego kodu w Julia (np. własnych funkcji)

6 Plan działania

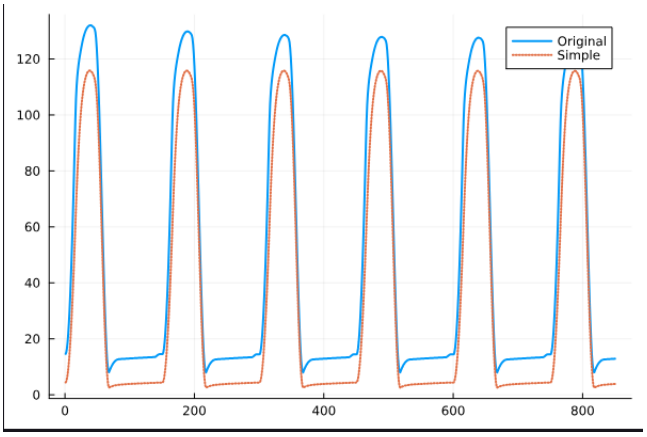
6.1 Trenowanie modelu PINN

Najważniejszym zadaniem jest wytrenowanie modelu PINN, który będzie lepiej odzworowywał rzeczywiste dane, niż uproszczony jednokomorowy model. Do procesu treningu jako dane referencyjne wykorzystuje się dane wygenerowane z modelu czterokomorowego. Kolejne kroki opracowania ulepszonego modelu to:

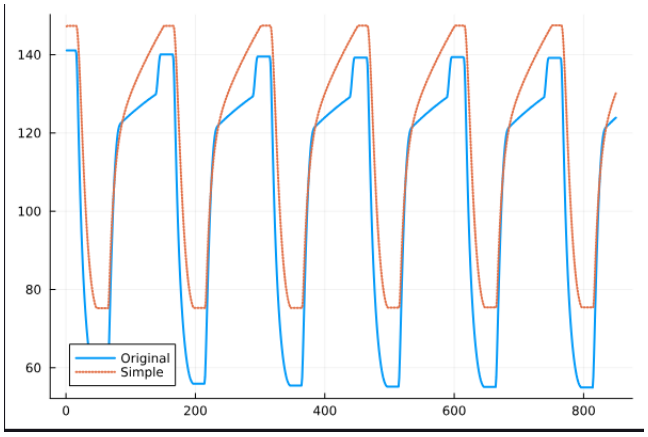
1. Przygotowanie zbioru treningowego i testowego: jako, że parametry opisujące zachowanie układu krążeniowo-naczyniowego w czasie są okresowe, do treningu wykorzystuje się fragment jednego okresu, a do testowania porównuje się ekstrapolację na dalszy przedział czasowy.

2. Aby model lepiej radził sobie z uogólnianiem, to warto dodać szum (np. generowany z rozkładu normalnego) do zbioru treningowego

3. Trenowanie modelu wzbogaconego o sieć neuronową (fully-connected, wiele możliwych architektur sieci)



Rysunek 2: Zmiana ciśnienia w lewej komorze serca



Rysunek 3: Zmiana objętości lewej komory serca w czasie

6.2 Regresja symboliczna

Po etapie uczenia, sieć neuronowa zostaje zredukowana z przestrzeni o wysokiej liczbie parametrów do postaci wyrażeń matematycznych. Ten krok prowadzi do powstania częściowo wyuczonego modelu, który jest bardziej interpretowalny niż standardowy model PINN(co jest kluczowe w dziedzinie medycyny) i często pomaga również w osiągnięciu dokładniejszych ekstrapolacji.

7 Prototyp

Prototyp produktu polega na zaprezentowaniu docelowego przypadku użycia. Na obecną chwilę planowa funkcjonalność obejmuje wykorzystanie wcześniej wytrenowanego modelu. Model po treningu zapisujemy przy użyciu biblioteki JLD2, do specjalnego formatu .jld2, która umożliwia łatwe zapisywanie i wczytywanie uprzednio wyliczonych wag. Wykorzystujemy również zestaw kilku równań różniczkowych, definiujących prosty i nieskomplikowany model, ale równocześnie mniej informatywny. Na samym końcu dołączamy do niego dane z wytrenowanej sieci, dzięki której jesteśmy w stanie zamodelować bardziej skomplikowane zależności między modelami.

7.1 Podstawowy kod treningu PINN

```
using OrdinaryDiffEq
using Lux, Zygote, Statistics, StableRNGs, ComponentArrays
using Optimization, OptimizationOptimisers, OptimizationOptimJL
using Optim, Measures, BenchmarkTools
using ForwardDiff

include("simple_CVS_model.jl")
using .simpleModel

rng = StableRNG(5958)

# Training range
tspan = (0.0, 7.0)
num_of_samples = 300
tsteps = range(5.0, 7.0, length = num_of_samples)

loaded_data = readdlm("data/original_data.txt")
original_data = Array{Float64}(loaded_data)

u0 = [6.0, 6.0, 6.0, 200.0, 0.0, 0.0, 0.0]
params = [0.3, 0.45, 0.006, 0.033, 1.11, 1.13, 11.0, 1.5, 0.03]

# Shi timing parameters
E_shift = 0.0
E_min = 0.03

tau_es = 0.3
tau_ep = 0.45
E_max = 1.5
R_mv = 0.006
tau = 1.0

NN = Lux.Chain(
    Lux.Dense(7, 24, sin),
    Lux.Dense(24, 24, sin),
```

```
    Lux.Dense(24, 7)
)

p, st = Lux.setup(rng, NN)
p = 0.5 * ComponentVector{Float64}(p)

function NIK_PINN!(du, u, p, t)
    p_lv, p_sa, p_sv, V_lv, q_av, q_mv, q_s = u
    tau_es, tau_ep, R_mv, Z_ao, R_s, C_sa, C_sv, E_max, E_min = params

    # Neural Network correction
    NN_output = NN(u, p, st)[1]

    elastance = ShiElastance(t, E_min, E_max, tau, tau_es, tau_ep, E_shift)
    elastance_deriv = DShiElastance(t, E_min, E_max, tau, tau_es, tau_ep,
↪   E_shift)

    du[1] = (q_mv - q_av) * elastance + p_lv / elastance * elastance_deriv +
↪   NN_output[1]
    du[2] = (q_av - q_s) / C_sa + NN_output[2]
    du[3] = (q_s - q_mv) / C_sv + NN_output[3]
    du[4] = q_mv - q_av + NN_output[4]
    du[5] = Valve(Z_ao, (du[1] - du[2]), p_lv - p_sa) + NN_output[5]
    du[6] = Valve(R_mv, (du[3] - du[1]), p_sv - p_lv) + NN_output[6]
    du[7] = (du[2] - du[3]) / R_s + NN_output[7]
end

prob_NN = ODEProblem(NIK_PINN!, u0, tspan, p)
s = solve(prob_NN, Vern7(), dtmax=1e-2, saveat=tsteps, reltol=1e-7,
↪   abstol=1e-4)

function predict(p)
    temp_prob = remake(prob_NN, p=p)
    temp_sol = solve(temp_prob, Vern7(), dtmax=1e-2, saveat=tsteps,
↪   reltol=1e-7, abstol=1e-4)
    return temp_sol
end

function loss(p)
    pred = predict(p)
    pred_vals = hcat(
        identity.(pred[1, :]),
        identity.(pred[2, :]),
        identity.(pred[3, :]),
        identity.(pred[4, :]),
        identity.(pred[5, :]),
        identity.(pred[6, :]),
```

```
        identity.(pred[7, :])
    )

    # Ground truth
    p_lv_true = original_data[:, 1]
    p_sa_true = original_data[:, 2]
    p_sv_true = original_data[:, 3]

    p_lv_pred = pred_vals[:, 1]
    p_sa_pred = pred_vals[:, 2]
    p_sv_pred = pred_vals[:, 3]
    q_av_pred = pred_vals[:, 5]
    q_mv_pred = pred_vals[:, 6]
    q_s_pred = pred_vals[:, 7]

    loss_data = mean(abs2, p_lv_pred .- p_lv_true) +
                mean(abs2, p_sa_pred .- p_sa_true) +
                mean(abs2, p_sv_pred .- p_sv_true)

    dt = step(tsteps)
    ts = collect(tsteps)
    dp_lv_dt(t) = s(t, Val{1}; idxs=1)
    dp_sa_dt(t) = s(t, Val{1}; idxs=2)
    dp_sv_dt(t) = s(t, Val{1}; idxs=3)

    dp_lv_vals = [dp_lv_dt(t) for t in ts]
    dp_sa_vals = [dp_sa_dt(t) for t in ts]
    dp_sv_vals = [dp_sv_dt(t) for t in ts]

    dp_lv_short = dp_lv_vals[1:end-1]
    dp_sa_short = dp_sa_vals[1:end-1]
    dp_sv_short = dp_sv_vals[1:end-1]

    q_av_short = q_av_pred[1:end-1]
    q_mv_short = q_mv_pred[1:end-1]
    q_s_short = q_s_pred[1:end-1]
    p_lv_short = p_lv_pred[1:end-1]
    p_sa_short = p_sa_pred[1:end-1]
    p_sv_short = p_sv_pred[1:end-1]

    physics_res_q_av = q_av_short .- Valve.(params[4], (dp_lv_short .-
↪   dp_sa_short), p_lv_short .- p_sa_short)
    physics_res_q_mv = q_mv_short .- Valve.(params[3], (dp_sv_short .-
↪   dp_lv_short), p_sv_short .- p_lv_short)
    physics_res_q_s = q_s_short .- ((dp_sa_short .- dp_sv_short) ./ params[5])

    loss_physics = mean(abs2, physics_res_q_av) +
```

```
                    mean(abs2, physics_res_q_mv) +
                    mean(abs2, physics_res_q_s)

    alpha = 0.01
    return loss_data + alpha * loss_physics
end

losses = Float64[]

callback = function (p, l)
    push!(losses, l)
    println("Current loss after $(length(losses)) iterations: $(losses[end])")
    return false
end

adtype = Optimization.AutoForwardDiff()
optf = Optimization.OptimizationFunction((x, p) -> loss(x), adtype)
optprob = Optimization.OptimizationProblem(optf, ComponentVector{Float64}(p))

# Train
w1 = Optimization.solve(optprob, ADAM(0.01), callback=callback, maxiters=500)
optprob1 = Optimization.OptimizationProblem(optf, w1.u)
PINN_sol = Optimization.solve(optprob1, ADAM(0.0001), callback=callback,
↪  maxiters=100)

prediction = predict(PINN_sol.u)

println(size(prediction))
println(size(prediction[1, :]))

data_to_save = hcat(
    prediction[1, :],
    prediction[2, :],
    prediction[3, :],
    prediction[4, :],
    prediction[5, :],
    prediction[6, :],
    prediction[7, :]
)

writedlm("data/baseline_pinn_data.txt", data_to_save)
println("Data from training range saved to baseline_pinn_data.txt")
```

8 Plan na następne miesiące

1. Udoskonalenie bazowego modelu PINN (lepsza generalizacja, większa dokładność)
2. Wprowadzenie większej transparentności funkcji i obliczeń, poprzez wprowadzenie regresji symbolicznej, dzięki której będziemy w stanie lepiej i łatwiej rozumieć zależności wynikające z zastosowania bardziej pojemnego modelu
3. Zwiększenie ilości dostepnych modeli, technik
4. Wprowadzenie szerszej funkcjonalności użytkownika do analizy konkretnego przypadku parametrów pacjenta

9 Literatura

1. *Review of Zero-D and 1-D Models of Blood Flow in the Cardiovascular System* by Yubing Shi, Patricia Lawford, and Rodney Hose
2. *A Hybrid Neural Ordinary Differential Equation Model of the Cardiovascular System* by Gevik Grigorian, Sandip V. George, Sam Lishak, Rebecca J. Shipley, and Simon Arridge
3. *Parameter Estimation for Closed-Loop Lumped Parameter Models of the Systemic Circulation Using Synthetic Data* by Nikolai L. Bjørdalsbakke, Jacob T. Sturdy, David R. Hose, and Leif R. Hellevik