SVEUČILIŠTE U SPLITU

Rješavanje prisilnog harmoničkog oscilatora metodom neuronskih mreža

1 UVOD

Prisilni prigušeni oscilator je fizikalni sustav koji oscilira pod utjecajem vanjske periodične sile, pri čemu su na sustav istovremeno prisutni i elastična sila te sila prigušenja. Tipičan primjer predstavlja masa m vezana na oprugu s konstantom k, na koju djeluje prigušena sila proporcionalna brzini gibanja ($-c\frac{dx}{dt}$), te vanjska pobudna sila harmonijskog tipa ($F(t) = F_0\cos(\omega t + \varphi_d)$).

1.1 MATEMATIČKI MODEL

Matematički model temelji se na Newtonovom drugom zakonu i opisuje se diferencijalnom jednadžbom drugog reda:

$$m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + kx = F_0\cos(\omega t + \varphi_d). \tag{1.1}$$

Ova jednadžba opisuje ravnotežu između tromosti mase, elastične sile, disipacije energije uslijed prigušenja i periodičnog vanjskog pobuđivanja. Rješenje jednadžbe sastoji se od dvaju dijelova:

1.2 PRIJELAZO RJEŠENJE (TRANSIENT)

Prijelazne oscilacije predstavljaju slobodno gibanje sustava koje se s vremenom prigušuje zbog otpora medija ili disipacije energije. Njegov oblik ovisi o početnim uvjetima (početni položaj i brzina mase). Prijelazno rješenje se dobije iz homogenog dijela diferencijalne jed. (1.1)

$$x_{\text{transient}}(t) = A_h e^{-\gamma t}\sin(\omega' t + \varphi_h), \tag{1.2}$$

gdje je koeficijent prigušenja

$$\gamma = \frac{c}{2m}, \tag{1.3}$$

a prigušena i vlastita frekvencija

$$\omega' = \sqrt{\omega_0^2 - \gamma^2}, \qquad \omega_0 = \sqrt{\frac{k}{m}}. \tag{1.4}$$

1.3 Stacionarno rješenje (steady-state)

Stacionarne oscilacije odgovaraju prisilnim oscilacijama koje se uspostavljaju pod utjecajem vanjske sile. Nakon što prođu prijelazne oscilacije, oscilator se približava maksimalnoj amplitudi i frekvenciji pobude iz izraza (1.6). Rješenje ovisi o frekvenciji pobude, konstanti gušenja i vlastitoj frekvenciji:

$$x_{\text{steady}}(t) = A\cos(\omega t - \varphi), \tag{1.5}$$

gdje amplituda i fazni pomak iznose

$$A = \frac{F_0/m}{\sqrt{(\omega_0^2 - \omega^2)^2 + 4\gamma^2\omega^2}}, \tag{1.6}$$

$$\varphi = \tan^{-1}\left(\frac{c\omega}{k - m\omega^2}\right) - \varphi_d. \tag{1.7}$$

1.4 Ukupno rješenje

Ukupno rješenje zapisuje se kao

$$x(t) = x_{\text{transient}}(t) + x_{\text{steady}}(t). \tag{1.8}$$

2 Fizikalna interpretacija

U početku gibanja sustav oscilira kombinacijom vlastitih i prisilnih oscilacija. Zbog prigušenja prolazni dio postupno nestaje, a nakon dovoljno dugog vremena u sustavu ostaju samo prisilne oscilacije iste frekvencije kao i vanjska sila. Amplituda i fazni pomak tih oscilacija određeni su parametrima sustava i frekvencijom pobude.

Posebno važan fenomen je **rezonancija**, kada frekvencija vanjske sile ω približno odgovara vlastitoj frekvenciji sustava ω_0. U tom slučaju amplituda stacionarnog rješenja doseže maksimalne vrijednosti, a rezonancijska krivulja snažno ovisi o jačini prigušenja.

3 Definirajmo Problem

Problem koji ćemo rješavati glasi

$$\frac{1}{2}\frac{d^2x}{dt^2} + 1\frac{dx}{dt} + 50x = 50\cos(10t). \tag{3.1}$$

Uz početne uvjete $x(t=0) = 0$ i $v(t=0) = 0$. Egzaktno rješenje se dobije rješavanjem jednadžbe metodom neodređenih koeficijenata, slijedi rješenje

$$x(t) = 5\sin(10t) - \frac{50}{3\sqrt{11}}\sin(3\sqrt{11}\,t)e^{-t} \tag{3.2}$$

4 Ukratko o neuronskim mrežama

Physics-Informed Neural Network (PINN) je neuronska mreža koja se trenira tako da poštuje zakone fizike izražene diferencijalnim jednadžbama. Umjesto da uči samo iz podataka, PINN u funkciju gubitka uključuje:

- početne i rubne uvjete (inicijalne i granične uvjete),
- rezidual diferencijalne jednadžbe, tj. odstupanje od zadanog fizikalnog zakona.

Physics-Informed Neural Network (PINN) funkcionira kao aproksimacija rješenja diferencijalne jednadžbe pomoću neuronske mreže. Neka su $\theta = \{W, b\}$ parametri (težine i biasi) mreže, a $u_{\text{PINN}}(t;\theta)$ izlaz mreže za ulaz t.

4.1 Loss funkcija

Funkcija gubitka sastoji se od više članova:

$$\mathcal{L}(t_i;\theta) = \underbrace{\left(u_{\text{PINN}}(0;\theta) - x_0\right)^2}_{\text{inicijalni uvjet (pozicija)}} \tag{4.1}$$

$$+ \underbrace{\lambda_1\left(\frac{du_{\text{PINN}}}{dt}(0;\theta) - v_0\right)^2}_{\text{inicijalni uvjet (brzina)}} + \underbrace{\frac{\lambda_2}{N}\sum_{i=1}^{N}\left(\left[m\frac{d^2}{dt^2} + \mu\frac{d}{dt} + k - F_0\cos(\omega t)\right]u_{\text{PINN}}(t_i;\theta)\right)^2}_{\text{fizikalni rezidual}}. \tag{4.2}$$

4.2 Minimizacija

Cilj je pronaći nove parametre θ koji minimiziraju Loss funkciju $\mathcal{L}(\theta)$:

$$\min_{\theta}\mathcal{L}(\theta).$$

U našem slučaju prisilnog oscilatora globalni minimum je nula.

4.3 POSTUPAK UČENJA

Trening mreže odvija se kroz sljedeće korake:

1. **Forward pass:** mreža računa $u_{\text{PINN}}(t;\theta)$ za odabrane točke t.
2. **Evaluacija loss funkcije:** izračunava se $\mathscr{L}(\theta)$.
3. **Backpropagation:** računa se gradijent $\nabla_\theta \mathscr{L}(\theta)$ korištenjem automatske diferencijacije, pri čemu se računaju derivacije po težinama i biasima θ
4. **Ažuriranje težina:** parametri mreže se ažuriraju pomoću vektora gradijenta loss funkcije koji je usmjeren u smjeru najbržeg pada funkcije

   $$\theta \leftarrow \theta - \eta \nabla_\theta \mathscr{L}(\theta),$$

   gdje je η stopa učenja (learning rate).

4.4 INTUICIJA

Ako mreža daje netočno rješenje, rezidual jednadžbe je velik, pa je i gubitak $\mathscr{L}(\theta)$ velik. Backpropagation tada "gura" parametre θ u smjeru u kojem mreža bolje zadovoljava diferencijalnu jednadžbu i početne uvjete. Nakon dovoljnog broja iteracija, $u_{\text{PINN}}(t;\theta)$ aproksimira analitičko rješenje zadane jednadžbe.

Razmotrimo diferencijalnu jednadžbu prinudnog titraja:

$$\frac{1}{2}\frac{d^2x}{dt^2} + \frac{dx}{dt} + 50x = 50\cos(10t), \tag{4.3}$$

uz početne uvjete

$$x(0) = x_0 = 0, \quad \frac{dx}{dt}(0) = v_0 = 0. \tag{4.4}$$

PINN aproksimira rješenje $x(t;\theta)$ neuronskom mrežom s parametrima θ. Rezidual diferencijalne jednadžbe definira se kao:

$$r(t;\theta) = \tfrac{1}{2}\frac{d^2x(t;\theta)}{dt^2} + \frac{dx(t;\theta)}{dt} + 50x(t;\theta) - 50\cos(10t).$$

Funkcija gubitka (loss function) tada je:

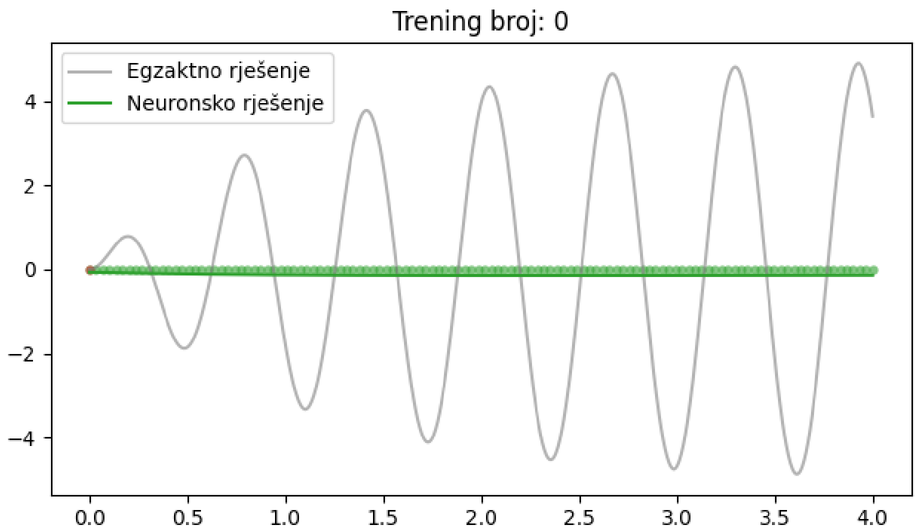
$$\mathscr{L}(t_i;\theta) = \big(u_{\text{PINN}}(t=0;\theta) - x_0\big)^2 \tag{4.5}$$

$$+\lambda_1\left(\frac{du_{\text{PINN}}}{dt}(t=0;\theta) - v_0\right)^2 + \frac{\lambda_2}{N}\sum_{i=1}^{N}\left(\left[m\frac{d^2}{dt^2} + \mu\frac{d}{dt} + k - F_0\cos(\omega t)\right]u_{\text{PINN}}(t_i;\theta)\right)^2. \tag{4.6}$$

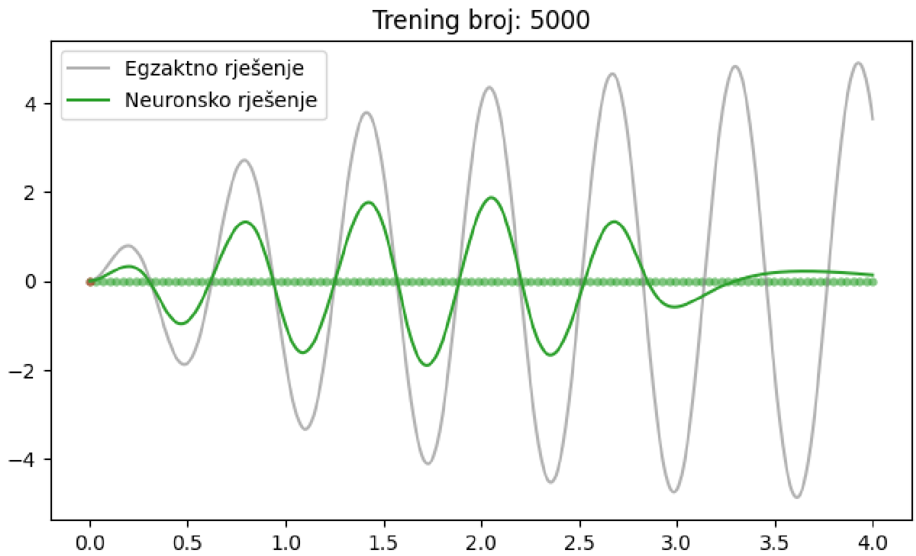
Na taj način, mreža uči rješenje ove diferencijalne jednadžbe kombinirajući zadane početne uvjete i zakon gibanja.

5 Rezultati simulacija prisilnog oscilatora

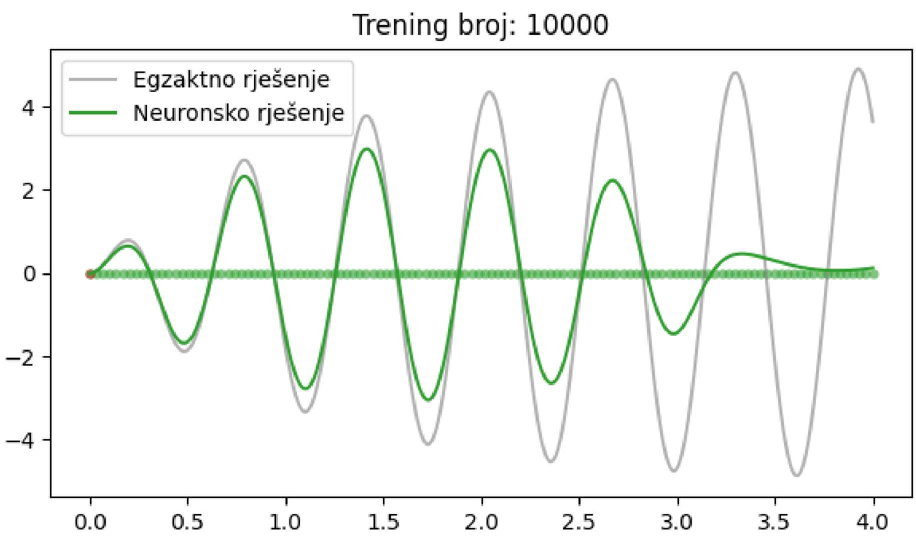
U ovom poglavlju prezentiramo numeričke rezultate simulacija diferencijalne jednadžbe prisilnog oscilatora iz metode neuralnih mreža za različite korake treninga. Trenirali smo mrežu u rasponu od 0 do 20000 treninga, a svaku pet tisućitu smo plotali. Vidimo dobro poklapanje između egzaktnog rješenja i numeričkog rješenja iz neuronskih mreža. Prijelazno rješenje traje do 3.5 sekunde i nakon toga oscilator poprima maksimalnu i konstantnu amplitudu od 5 m.



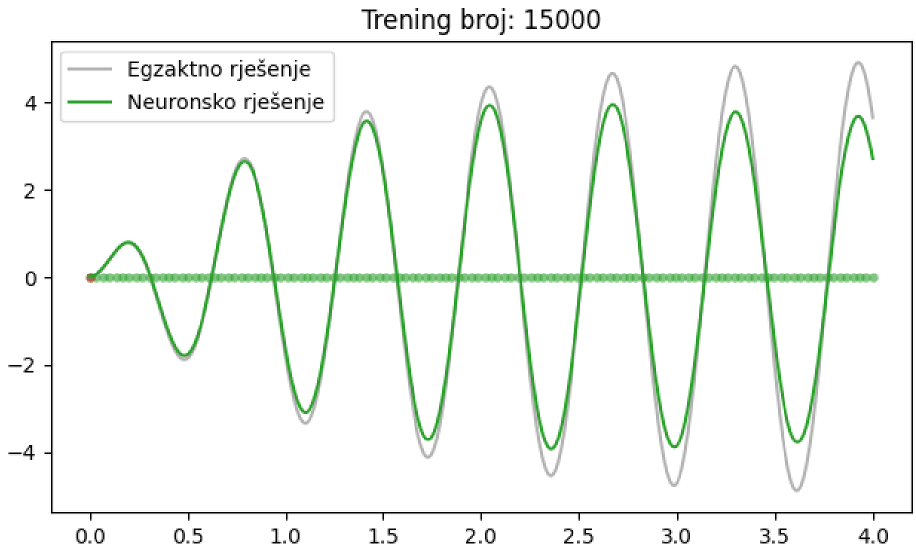
Slika 5.1: početna inicijalizacija težina i biasa



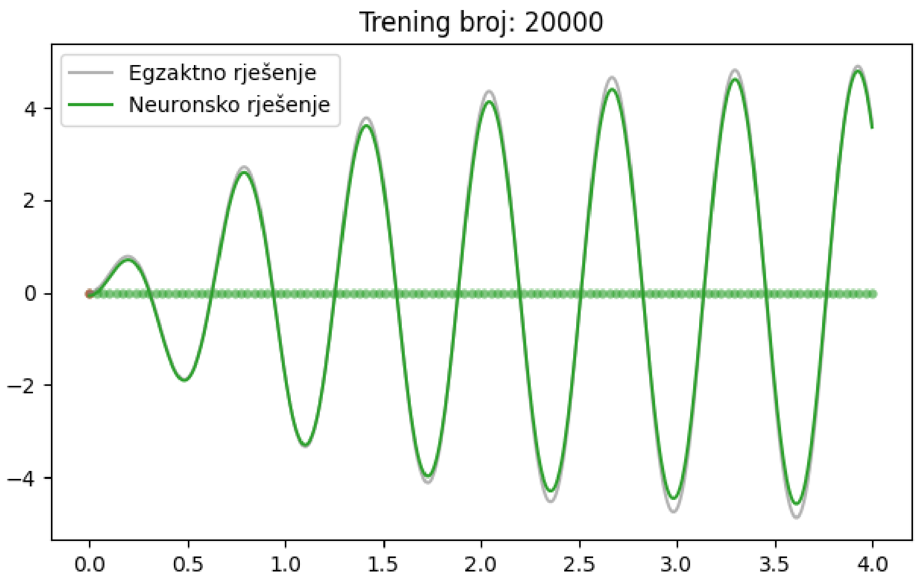
Slika 5.2: trening 5000



Slika 5.3: trening 10 000



Slika 5.4: trening 15 000



Slika 5.5: trening 20 000

6 KOD

```
import torch
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt

def exact_solution(d, w0, t):
    #Analiticko rjesenje prisilnog gusenoga oscilatora
    assert d < w0
    wg = np.sqrt(w0**2 - d**2)
    # frekvencija gusenja, d koeficijent gusenja gamma
    A = -50/(3*np.sqrt(11))
    hom = torch.sin(wg*t)
    part=5*torch.sin(w0*t)
    exp = torch.exp(-d*t)
    u = exp*A*hom+part
    return u


class FCN(nn.Module):
    #definira potpuno spojenu neuronsku mrezu

    def __init__(self, N_INPUT, N_OUTPUT, N_HIDDEN, N_LAYERS):
        super().__init__()
        activation = nn.Tanh
        self.fcs = nn.Sequential(*[
            nn.Linear(N_INPUT, N_HIDDEN),
            activation()])
        self.fch = nn.Sequential(*[
                        nn.Sequential(*[
                        nn.Linear(N_HIDDEN, N_HIDDEN),
                        activation()]) for _ in range(N_LAYERS - 1)])
        self.fce = nn.Linear(N_HIDDEN, N_OUTPUT)

    def forward(self, x):
        x = self.fcs(x)
        x = self.fch(x)
        x = self.fce(x)
        return x


# stvori neuronsku mrezu
pinn = FCN(1, 1, 120, 3)

# definiraj pocetne tocke treniranja
t_boundary = torch.tensor(0.).view(-1, 1).requires_grad_(True)

# definiraj tocke treniranja za domenu diferencijalne jed. prisilnog
                                            oscilatora
t_physics = torch.linspace(0, 4, 120).view(-1, 1).requires_grad_(True)
```

```
# treniraj mrezu
mu, k = 1, 50
d, w0 = mu/(2*0.5), 10
t_test = torch.linspace(0, 4, 800).view(-1, 1)
u_exact = exact_solution(d, w0, t_test)
optimiser = torch.optim.Adam(pinn.parameters(), lr=1e-3)

for i in range(20001):
    optimiser.zero_grad()

    # hiperparametri regulacije clanova
    lambda1, lambda2 = 1e-1, 1e-3

    # racunaj  gresku ili loss za po etnu tocku
    u = pinn(t_boundary)  # (1,1)
    loss1 = (torch.squeeze(u))**2

    dudt = torch.autograd.grad(u, t_boundary, torch.ones_like(u),
                                         create_graph=True)[0]
    loss2 = (torch.squeeze(dudt))**2

    # racunaj loss za fiziklni clan (diferencijalnu jed. prisilnog
                                         oscilatora)
    u = pinn(t_physics)
    dudt = torch.autograd.grad(u, t_physics, torch.ones_like(u),
                                         create_graph=True)[0]
    d2udt2 = torch.autograd.grad(dudt, t_physics, torch.ones_like(dudt),
                                         create_graph=True)[0]
    force=-50*torch.cos(w0*t_physics)
    loss3 = torch.mean((0.5*d2udt2 + mu*dudt + k*u+force)**2)

    # algoritam backpropagation ili gradijentni spust (azuriraj tezine za
                                          svaku vezu) i optimiziraj
    loss = loss1 + lambda1*loss2 + lambda2*loss3
    loss.backward()
    optimiser.step()

    # plotaj graf i prati kako treniranje napreduje i greska se
                                         minimizira
    if i % 5000 == 0:
        u = pinn(t_test).detach()
        plt.figure(figsize=(6, 2.5))
        plt.scatter(t_physics.detach()[:, 0],torch.zeros_like(t_physics)[
                                             :, 0],s=20, lw=0, color="tab
                                             :green", alpha=0.6)

        plt.scatter(t_boundary.detach()[:, 0],torch.zeros_like(t_boundary
                                             )[:, 0],s=20, lw=0, color="
                                             tab:red", alpha=0.6)

        plt.plot(t_test[:, 0], u_exact[:, 0], label="Egzaktno rje enje",
                                              color="tab:grey", alpha=0.6
                                             )
```

```
        plt.plot(t_test[:, 0], u[:, 0], label="Neuronsko rje enje",
                                                color="tab:green")
        plt.title(f"Trening broj: {i}")
        plt.legend()
        plt.show()
```