Neuronske mreže podržane fizičkim zakonima - Praktikum

Miloš Ivanović

2023.

Miloš Ivanović

Neuronske mreže podržane fizičkim zakonima
Praktikum

Recenzenti
prof. dr Boban Stojanović
doc. dr Višnja Simić

Izdavač
Prirodno-matematički fakultet Kragujevac
Radoja Domanovića 12
Kragujevac

Štampa
GRAFIKA GALEB DOO, NIŠ

Tiraž
70 primeraka

ISBN-978-86-6009-096-8

Sadržaj

1 Uvod **5**
- 1.1 Kratka istorija metode . . . 5
- 1.2 Neuronske mreže podržane fizičkim zakonima . . . 6
- 1.3 Primer konstrukcije funkcije gubitka . . . 9

2 Provođenje toplote **13**
- 2.1 Jednačina provođenja toplote . . . 13
- 2.2 Jednodimenzioni direktni problem . . . 14
- 2.3 Jednodimenzioni inverzni problem . . . 22
- 2.4 Stefanov problem u modelima topljenja . . . 24

3 Mehaničke oscilacije **33**
- 3.1 Oscilator . . . 33
- 3.2 Implementacija . . . 40
- 3.3 Rešenja direktnog problema . . . 44
- 3.4 Inverzni problem . . . 47

4 Hidrologija **53**
- 4.1 Uvod . . . 53
- 4.2 Strujanje podzemnih voda . . . 53
- 4.3 Propagacija talasa u otvorenom kanalu . . . 62

5 Akustika **73**
- 5.1 Uvod . . . 73

5.2 Rešavanje na domenu oblika kvadrata 74
5.3 Rešavanje na domenu oblika kvadrata sa šupljinom 80
5.4 Dvodimenzioni problem sa sočivom 85

6 Modelovanje mišića **93**
6.1 Uvod 93
6.2 Izometrijski slučaj 96

7 Modelovanje proizvodnje solarnih elektrana **103**
7.1 Uvod 103
7.2 Osnovne jednačine modela 113
7.3 Primer proračuna proizvodnje 115
7.4 Rezultati 121

Bibliografija **125**

Predgovor

Neuronske mreže podržane fizičkim zakonima (*Physics-Informed Neural Networks* - PINN) ili na srpskom jeziku NMPFZ su tip univerzalnih funkcija aproksimacije koje mogu da se treniraju tako da usvoje poznavanje bilo kog fizičkog zakona koji se može opisati parcijalnim diferencijalnim jednačinama, a koji važi u određenom prostorno-vremenskom domenu. Obuka ovog tipa neuronskih mreža se postavlja na taj način da poštuje simetrije, invarijantnost ili održavanje principa koji počivaju na fizičkim zakonima iskazanim u obliku parcijalnih diferencijalnih jednačina.

Obične duboke neuronske mreže nisu dovoljno robusne u većini slučajeva kada se vezuju za matematički iskazane zakone u biologiji, mehanici, elektrotehnici, itd. S druge strane, kod NMPFZ mreža, prethodno znanje opštih fizičkih zakona se u procesu treniranja neuronskih mreža postavlja kao regularizacioni agent koji ograničava prostor dozvoljenih rešenja, što povećava tačnost aproksimirane funkcije. Na ovaj način, ugrađivanjem fizičkih zakona opisanih parcijalnim diferencijalnim jednačinama u neuronsku mrežu dobijamo poboljšanje, što olakšava algoritmu učenja da dobije što tačnije rešenje i da dobro generalizuje, čak i sa veoma malom količinom tzv. kolokacionih tačaka.

Tradicionalno, za rešavanje (sistema) diferencijalnih jednačina numeričkim putem, već decenijama se koriste klasične metode, kao što su metoda konačnih razlika (eng. *Finite Difference Method*), metoda konačnih elemenata (eng. *Finite Element Method*), metoda konačnih zapremina (eng. *Finite Volume Method*), itd. Uprkos njihovoj popularnosti, sve ove metode imaju i određena ograničenja. Pre svih, tu su dva problema. Prvi je velika računska kompleksnost, a drugi asimilacija eksternih izvora podataka dobijenih merenjem na modelovanom sistemu. Takođe, rešavanje inverznih problema tj. pretraga nepoznatih

ili nesigurnih parametara klasičnim metodama postaje gotovo nemoguća za iole kompleksnije probleme iz inženjerske prakse. NMPFZ mreže nude jednostavne i fleksibilne mehanizme koji sve ove probleme uzimaju u obzir od početka, samim svojim konceptom.

Ovaj praktikum je nastao iz motivacije da se ovom relativno novom konceptu, koji je prvi put publikovan 2017. godine od strane Raissi et al.[1] posveti dužna pažnja na srpskom govornom području i time u određenoj meri snizi barijera za ulazak u ovu „hibridnu" oblast na granici između numeričkog modelovanja i mašinskog učenja. Približavanjem ove dve oblasti bi se, po mišljenju autora, puno toga dobilo. Fizičari i inženjeri bi dobili fleksibilan alat za približno rešavanje direktnih i inverznih problema koji im omogućava da brzo provere hipoteze, integrišu merenja i identifikuju parametre. S druge strane, istraživačima iz polja mašinskog učenja se nudi način da već postojeća znanja o fizičkim pojavama i inženjerskim zakonitostima na relativno jednostavan način integrišu u svoje, do sada čisto statističke, modele i time im poboljšaju pouzdanost i mogućnost predviđanja.

Zbirka se sastoji iz detaljno postavljenih i rešenih primera iz prakse autora i saradnika. Dotakli smo probleme iz oblasti klasične mehanike, provođenja toplote i neke jednostavnije hidrološke probleme. Takođe predlažemo načine za integraciju merenja i optimizaciju hiper-parametara poput broja slojeva i neurona u neuronskoj mreži, aktivacionih funkcija itd.

Autor se zahvaljuje saradnicima Centra za računarsko modelovanje i optimizaciju (CERAMO) Prirodno-matematičkog fakulteta u Kragujevcu[2]. Pre svega, zahvalnost dugujem dr Bobanu Stojanoviću, redovnom profesoru PMF-a, rukovodiocu više projekata u kojima je primenu našla metoda koja je tema ovog praktikuma. Mladi kolega Filip Bojović doprineo je uspešnoj implementaciji problema *Propagacija poplavnog talasa u otvorenom kanalu*, a koleginica Branka Andrijević u problemu *Filtracija podzemnih voda* i *Modelovanje proizvodnje solarnih panela*. Bogdan Milićević je značajno doprineo rešavanju problema modelovanja mišića pomoću NMPFZ, a odgovoran je i za srpski prevod naziva ove metode, na čemu mu se zahvaljujem. Koleginici Višnji Simić dugujem zahvalnost za pažljivi pregled teksta i korisne terminološke sugestije.

Na kraju, ali ne i najmanje važno, zahvalnost dugujem kolegi Vladimiru Milovanoviću sa Fakulteta inženjerskih nauka u Kragujevcu, koji mi je ukazao da savremeni nastavni materijal treba da bude otvoren i da je danas veoma važno da bude dostupan i na vebu i kao klasično štampano izdanje. Kompletan materijal, uključujući izvorne tekstove, slike, izvorni kod primera, kao i uputstva za njihovo pokretanje mogu se pronaći na autorovom Guthub repozitorijumu[3]. Ukoliko čitalac uoči bilo kakvu grešku ili nedoslednost i

[1] https://maziarraissi.github.io/PINNs/

[2] https://www.pmf.kg.ac.rs/

[3] https://github.com/imilos/pinn-skripta

ima volje i želje, kontakt putem odeljka *Issues* ili, još bolje, *Pull Request* su više nego dobrodošli.

U Kragujevcu,
decembar 2022. godine

– autor

1 Uvod

1.1 Kratka istorija metode

Tokom poslednje decenije se ubrzano razvijaju različite metode dubokog učenja rešavanje različitih vrsta problema iz oblasti veštačke inteligencije, kao što su prepoznavanje slike, prepoznavanje govora, obrada prirodnog jezika (*Natural Language Processing* - NLP), pretraživanje, sistemi za preporuke, bioinformatika, itd. Međutim, klasično nadgledano duboko učenje kao metoda nije pogodno za rešavanje baš svih vrsta problema, bez obzira na dovoljnu količinu dostupnih podataka koji opisuju ponašanje modelovanog sistema. Na primer, problemi opisani kroz linearne i nelinearne jednačine i sisteme jednačina nikad i nisu bili u fokusu dubokog učenja. Klasične numeričke metode i dalje drže apsolutni primat u numeričkom rešavanju parcijalnih diferencijalnih jednačina. Metode kao što su konačne razlike (*Finite Difference Method*), konačne zapremine (*Finite Volume Method*) i konačni elementi (*Finite Element Method*) se i dalje smatraju najsavremenijim metodama zbog njihove robusnosti, efikasnosti i mogućnosti primene u širokom spektru problema.

Sa druge strane, rešavanje nelinearnih inverznih problema klasičnim numeričkim metodama zahteva proračune koji su izuzetno računarski i vremenski zahtevni. Pretraga optimalnih parametra modela uključuje iterativni postupak koji često uključuje desetine i stotine prolaza direktne simulacije. Zbog toga se tačnost često žrtvuje za efikasnost, te ove vrste pretraga često nisu iscrpne i uključuju tzv. meta-heuristike, bez garancije da će se pronaći najbolje (optimalno) rešenje. Primeri iz prakse autora i saradnika najčešće koriste genetske algoritme (*Genetic Algorithm* - GA), sprovode optimizaciju baziranu na simulaciji (*Simulation Based Optimization*) i zahtevaju upotrebu više stotina procesora da bi se do rešenja (koje nije garantovano optimalno) došlo u iole razumnom vremenskom

roku, prema Ivanovic *et al.* [ISS+15], Simic *et al.* [SSI19] i Ivanovic and Simic [IS22]. Pored kardinalnih poteškoća pri rešavanju inverznih problema, očigledni su još i sledeći nedostaci klasičnih numeričkih metoda:

- Nije moguće na lak način u model uključiti **podatke sa šumom**.
- **Generisanje proračunske mreže** ostaje složena manuelna operacija, često podložna greškama.
- Probleme koji uključuju **veći broj dimenzija** nije moguće rešiti u realnim vremenskim okvirima računarskim resursima kojima trenutno raspolažemo.

Da bi se eliminisali ovi nedostaci klasičnih numeričkih metoda, razvijena je nova metoda dubokog učenja za rešavanje parcijalnih diferencijalnih jednačina. Ta metoda, pod imenom **Neuronske mreže podržane fizičkim zakonima** (*Physics Informed Neural Networks* - PINN), koristi se za rešavanje problema nadgledanog učenja uz poštovanje bilo kog zakona fizike opisanog opštim nelinearnim parcijalnim diferencijalnim jednačinama Raissi *et al.* [RPK19].

Glavna inovacija Neuronskih mreža potkrepljenim fizičkim zakonima (NMPFZ) u poređenju sa klasičnim dubokim neuronskim mrežama je uvođenje funkcije gubitka koja kodira osnovne jednačine fizičkih zakona, uzima izlaz duboke mreže, koja se zove aproksimator, i izračunava vrednost gubitka Markidis [Mar21]. Funkcija gubitka koja se odnosi na diferencijalnu jednačinu se minimizira obukom aproksimatorske neuronske mreže, gde se diferencijalni operatori primenjuju korišćenjem automatske diferencijacije. Automatska diferencijacija je preduslov da bi se uopšte obavilo treniranje propagacijom unazad i poseduju je sve biblioteke za mašinsko učenje, kao što su *Tensorflow*, *PyTorch*, *Theano*, i druge.

1.2 Neuronske mreže podržane fizičkim zakonima

Neuronska mreža potkrepljena fizičkim zakonima (u daljem tekstu NMPFZ) je tehnika mašinskog učenja koja može se koristiti za aproksimaciju rešenja parcijalne diferencijalne jednačine. Parcijalne diferencijalne jednačine sa odgovarajućim početnim i graničnim uslovima mogu se izraziti u opštem obliku kao:

$$\begin{aligned} u_t + \mathcal{N}[u] &= 0, \quad X \in \Omega,\ t \in [0,T], \\ u(X,0) &= h(X), \quad X \in \Omega, \\ u(X,t) &= g(X,t), \quad X \in \Omega_g,\ t \in [0,T]. \end{aligned} \tag{1.1}$$

Ovde je $\mathcal{N}$ diferencijalni operator, $X \in \Omega \subseteq R^d$ i $t \in R$ predstavljaju prostorne i vremenske koordinate, respektivno, dok je $\Omega \subseteq R$ celokupni domen problema. $\Omega_g \subseteq \Omega$ predstavlja računski domen graničnih uslova, $u(X,t)$ je rešenje parcijalne diferencijalne jednačine sa početnim uslovom $h(X)$ i graničnim uslovom $g(X,t)$. Ovakva formulacija se takođe može primeniti i na parcijalne diferencijalne jednačine višeg reda, pošto se jednačine višeg reda mogu napisati i u obliku sistema jednačina prvog reda.

U originalnoj formulaciji Raissi *et al.* [RPK19], NMPFZ se sastoji od dve podmreže:

- aproksimator mreže i
- rezidualne mreže.

Aproksimator mreža prima ulaz (X,t), prolazi kroz proces obuke i kao izlaz daje približno rešenje $\widehat{u}(X,t)$ parcijalne diferencijalne jednačine. Mreža aproksimatora se trenira na mreži tačaka, tzv. **kolokacionih tačaka**, uzorkovanih iz domena problema. Težine i pristrasnosti (eng. *bias*) aproksimator mreže su parametri koji se mogu trenirati minimiziranjem kompozitne funkcije gubitka u sledećem obliku:

$$\mathcal{L} = \mathcal{L}_r + \mathcal{L}_0 + \mathcal{L}_b, \tag{1.2}$$

gde su

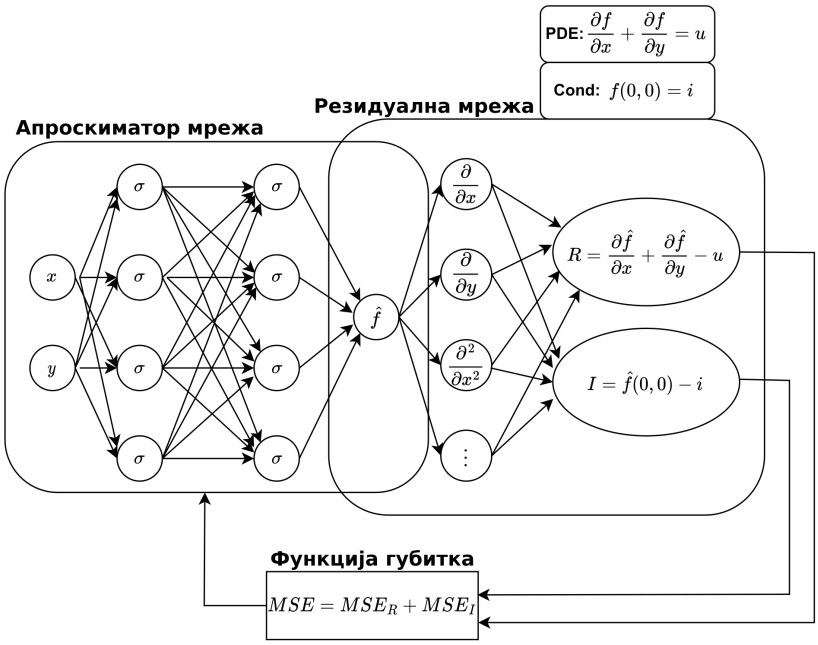
$$\begin{aligned} \mathcal{L}_r &= \frac{1}{N_r}\sum_{i=1}^{N_r} \left| u\left(X^i,t^i\right) + \mathcal{N}\left[u\left(X^i,t^i\right)\right]\right|^2, \\ \mathcal{L}_0 &= \frac{1}{N_0}\sum_{i=1}^{N_0} \left| u\left(X^i,t^i\right) - h^i\right|^2, \\ \mathcal{L}_b &= \frac{1}{N_b}\sum_{i=1}^{N_b} \left| u\left(X^i,t^i\right) - g^i\right|^2. \end{aligned} \tag{1.3}$$

Ovde, $\mathcal{L}_r$, $\mathcal{L}_0$ i $\mathcal{L}_b$ predstavljaju reziduale osnovne diferencijalne jednačine, početnih i graničnih uslova, respektivno. Pored toga, N_r, N_0 i N_b su brojevi kolokacionih tačaka domena problema, domena početnih i graničnih uslova, respektivno. Ovi reziduali se izračunavaju komponentom NMPFZ modela koji se ne obučava, a zove se **rezidualna mreža**. Da bi se izračunao gubitak $\mathcal{L}_r$, NMPFZ zahteva izvode izlaza u odnosu na ulaze. Taj račun se postiže tzv. automatskom diferencijacijom.

Automatska diferencijacija je ključni pokretač razvoja NMPFZ-a i ključni je element koji razlikuje NMPFZ od sličnih nastojanja 90-ih godina prošlog veka. Na primer, Psichogios and Ungar [PU92] i Lagaris *et al.* [LLF98] su se oslanjali na manuelno izvođenje

pravila propagacije unazad. Danas se računa na automatsku diferencijaciju koja je implementirana u većini okvira za duboko učenje, kao što su Tensorflow[4] i PyTorch[5]. Na ovaj način izbegavamo numeričku diskretizaciju tokom računanja izvoda svih redova u prostor-vremenu.

Šema tipičnog NMPFZ-a je prikazana na Sl. 1.1 na kojoj je jednostavna parcijalna diferencijalna jednačina $\frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} = 0$ iskorišćena kao primer. Kao što je prikazano, mreža aproksimatora se koristi za aproksimaciju rešenja $u(X, t)$, koje zatim ide na rezidualnu mrežu za izračunavanje funkcije gubitka diferencijalne jednačine $\mathcal{L}_r$, gubitka graničnih uslova $\mathcal{L}_b$, i gubitka početnih uslova $\mathcal{L}_0$. Težine i pristrasnosti aproksimatorske mreže obučeni su korišćenjem prilagođene funkcije gubitka koja se sastoji od reziduala $\mathcal{L}_r$, $\mathcal{L}_0$, i $\mathcal{L}_b$ kroz tehniku gradijenta spuštanja zasnovanu na propagaciji unazad.



Sl. 1.1: Arhitektura NMPFZ-a i standardna petlja za obuku NMPFZ-a konstruisana za rešavanje jednostavne parcijalne diferencijalne jednačine, gde *PDE* i *Cons* označavaju jednačine, dok *R* i *I* predstavljaju njihove reziduale. Mreža aproksimatora je podvrgnuta procesu obuke i daje približno rešenje. Rezidualna mreža je deo NMPFZ-a koji se ne obučava i koji je sposoban da izračuna izvode izlaza aproksimatorske mreže u odnosu na ulaze, što rezultira kompozitnom funkcijom gubitka, označenom sa *MSE*.

[4] https://www.tensorflow.org/
[5] https://pytorch.org/

U narednoj sekciji *Primer konstrukcije funkcije gubitka* (stranica 9) opisaćemo kako bi izgledala konstrukcija kompozitne funkcije gubitka za logističku jednačinu.

1.3 Primer konstrukcije funkcije gubitka

Rezonovanje iz prethodnog odeljka *Neuronske mreže podržane fizičkim zakonima* (stranica 6) ćemo potkrepiti jednostavnim primerom. Započnimo prvo tako što ćemo uzeti neku jednostavnu diferencijalnu jednačinu. Izabraćemo logističku jednačinu[6] (*Logistic Equation*) koja predstavlja model rasta populacije. Ta jednačina glasi:

$$\frac{df}{dt} = R \cdot t(1-t) \tag{1.4}$$

Funkcija $f(t)$ predstavlja stopu rasta populacije tokom vremena t, dok parametar R određuje maksimalnu stopu rasta. Kako bismo od familije krivih koje zadovoljavaju rešenje ove obične diferencijalne jednačine izabrali jednu krivu kao rešenje, moramo postaviti granični, tj. početni uslov. Neka to bude:

$$f(t=0) = \frac{1}{2}. \tag{1.5}$$

Kako je analitičko rešenje ove jednačine poznato, poređenjem sa njim možemo proveriti tačnost NMPFZ. Time ćemo početi da razotkrivamo potencijal primene i na kompleksnije obične i parcijalne diferencijalne jednačine. Da se podsetimo, NMPFZ su zasnovane na dve fundamentalne osobine neuronskih mreža:

- Formalno je pokazano da su **neuronske mreže univerzalne funkcije aproksimacije**, tako da neuronska mreža može da aproksimira bilo koju funkciju, a samim tim i rešenje za našu logističku jednačinu.
- Jednostavno je i jeftino izračunati izvode bilo kog reda izlaza iz neuronske mreže za bilo koji dati ulaz korišćenjem **automatske diferencijacije**.

Dakle, kao što je rečeno, možemo da konstruišemo funkciju gubitka tako da, kada se minimizuje, diferencijalna jednačine bude zadovoljena:

$$\mathcal{L}_r = \frac{df_{NN}(t)}{dt} - R \cdot t(1-t) = 0, \tag{1.6}$$

gde je $f_{NN}(t)$ izlaz neuronske mreže sa jednim ulazom čiji se izvod izračunava automatskim diferenciranjem. Odmah vidimo da, ukoliko izlaz iz mreže zadovoljava (1.6),

[6] https://en.wikipedia.org/wiki/Logistic_function

zapravo se ta jednačina rešava. Da bi se izračunao stvarni doprinos funkciji gubitka koji se dobija iz diferencijalne jednačine, potrebno je specificirati skup tzv. kolokacionih tačaka u domenu problema i proceniti srednju kvadratnu grešku (*Mean Squared Error - MSE*) ili neku drugu funkciju gubitka:

$$\mathcal{L}_r = \frac{1}{N_r} \sum_{i=1}^{N_r} \left(\frac{df_{NN}}{dt}\bigg|_{t_i} - Rt_j(1-t_i) \right)^2, \tag{1.7}$$

gde je N_r broj kolokacionih tačaka t_j u kojima se računa funkcija gubitka. Gubitak zasnovan samo na rezidualima ne osigurava jedinstveno rešenje, pa stoga uključujemo i granični uslov, tako što ga dodajemo funkciji gubitka na isti način kao u jednačini (1.7):

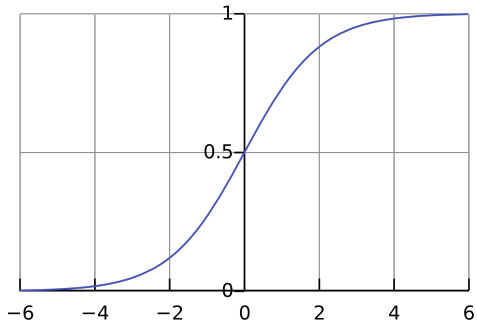
$$\mathcal{L}_0 = \frac{1}{N_0} \sum_{i=1}^{N_0} \left(f_{NN}(t)|_{t_i} - \frac{1}{2} \right)^2, \qquad t_i \approx 0. \tag{1.8}$$

Sada imamo oba elementa da definišemo ukupnu funkciju gubitka $\mathcal{L}$:

$$\mathcal{L} = \mathcal{L}_r + \mathcal{L}_0. \tag{1.9}$$

Tokom treniranja neuronske mreže, gornji izraz se minimizuje i izlaz iz mreže trenira da zadovolji diferencijalnu jednačinu i zadat granični uslov, čime se aproksimira konačno, jedinstveno rešenje diferencijalne jednačine. Koncept NMPFZ je veoma jednostavan, i koristeći ideju opisanu u prethodnom tekstu, možemo dodati više graničnih uslova, dodati kompleksnije ili rešavati vremenski zavisne višedimenzionalne probleme koristeći mrežu sa višestrukim ulazima.

Rešenje naše logističke jednačine je dobro poznata sigmoidna funkcija prikazana na Sl. 1.2.



Sl. 1.2: Sigmoidna funkcija koja se dobija kao rešenje jednačine (1.4) sa početnim uslovom (1.5)

1.3.1 Način izlaganja primera

Izlaganje ćemo nastaviti konkretnim primerima. Svako poglavlje praktikuma (Provođenje toplote, Oscilacije, …) bavi se posebnom fenomenologijom i sadrži po jedan ili više direktnih ili inverznih rešenih problema. Svaki primer je potkrepljen teorijskom pozadinom, pripadajućim programskim kodom koji implementira NMPFZ rešenje, kao i analizom tačnosti i efikasnosti NMPFZ rešenja u odnosu na analitička rešenja, ukoliko postoje, ili rešenja dobijena klasičnim numeričkim metodama.

Valja napomenuti da primeri nisu sortirani po težini, već isključivo po fenomenologiji koja se modeluje. Ako ipak treba da preporučimo čitaocu odakle da krene sa praktičnim radom, recimo da *Mehaničke oscilacije* (stranica 33) i *Jednodimenzioni direktni problem* (stranica 14) predstavljaju dobru osnovu.

2 Provođenje toplote

2.1 Jednačina provođenja toplote

U fizici, hemiji i inžinjerstvu, fenomeni prenosa se bave mehanizmima prenosa neke fizičke veličine sa jedne lokacije na drugu. Tri osnova mehanizma prenosa su provođenje (difuzija), prenošenje (konvekcija) i zračenje (radijacija). Tri osnovne veličine koje se tretiraju u fenomenima prenosa su:

- prenos toplote,
- prenos mase i
- prenos količine kretanja.

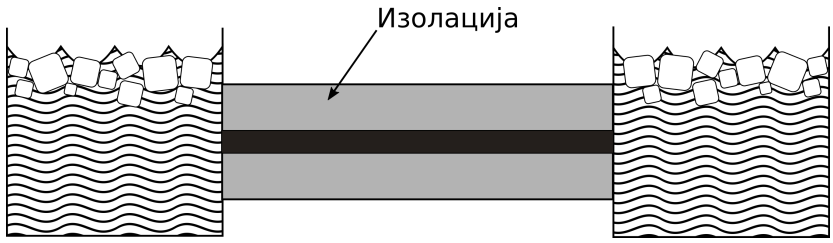
Sve tri veličine prenose se na sličan način. Ako govorimo o toploti, Furijeov zakon opisuje proporcionalnost fluksa toplote i gradijenta temperature. Koeficijent proporcionalnosti naziva se koeficijentom toplotne provodljivosti. Fenomen provođenja toplote u vremenu u jednoj prostornoj dimenziji definisan je sledećom jednačinom:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial t^2}, \qquad 0 < x < L,\, t \geq 0 \tag{2.1}$$

gde je $u(x,t)$ temperatura, a α konstantan koeficijent. U narednom odeljku *Jednodimenzioni direktni problem* (stranica 14) definisaćemo jednostavan problem, rešiti ga pomoću NMPFZ i uporediti dobijeno rešenje sa analitičkim izrazom.

2.2 Jednodimenzioni direktni problem

Tanak štap od homogenog materijala je okružen izolacijom, tako da se promene temperature u štapu dešavaju samo kao posledica razmene toplote ka krajevima štapa i provođenja toplote duž štapa. Štap je jedinične dužine. Oba kraja su izložena mešavini vode i leda temperature 0. Početna temperatura na rastojanju x od levog kraja štapa je $sin(\pi x)$, kao što se vidi na Sl. 2.1.

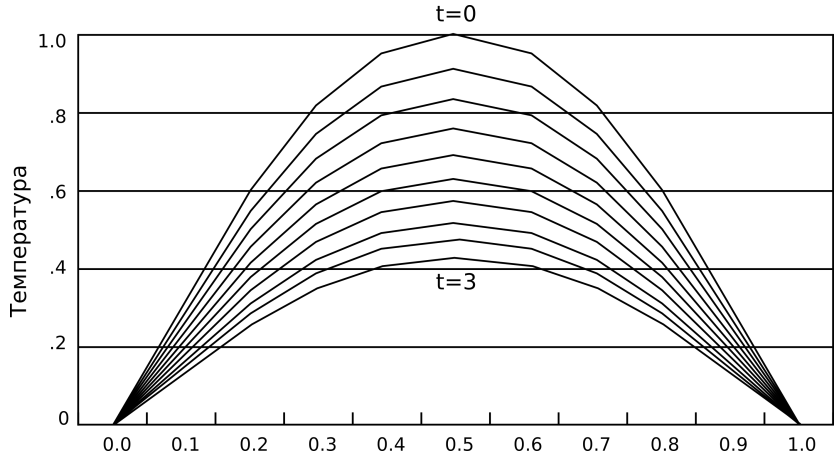


Sl. 2.1: Eksperimentalna postavka problema provođenja toplote duž štapa. Na krajevima štapa nalazi se mešavina vode i leda. Štap je izolovan od uticaja spoljašnje sredine.

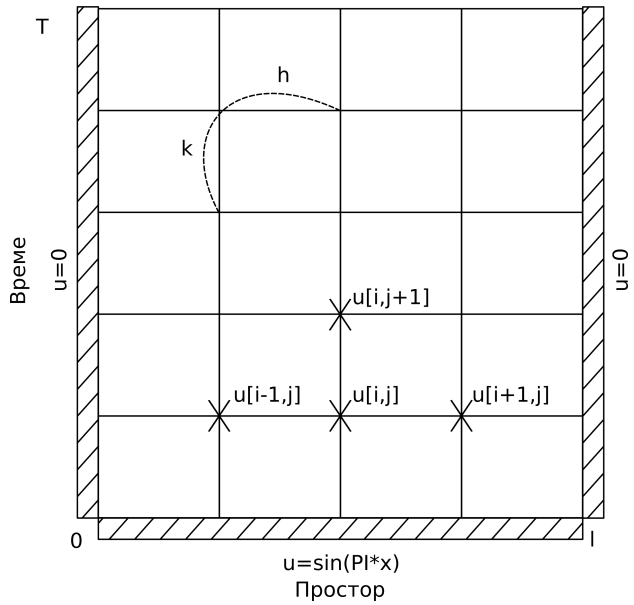
Radi poređenja, pokazaćemo sada kako se ovaj relativno jednostavan problem formuliše pomoću klasične metode konačnih razlika, a zatim ćemo rešiti direktni i inverzni problem opisane postavke NMPFZ metodom.

2.2.1 Rešavanje metodom konačnih razlika (MKR)

Parcijalna diferencijalna jednačina (2.1) modeluje temperaturu u bilo kojoj tački štapa u bilo kom vremenskom trenutku prema Recktenwald [Rec04]. Ova jednačina se rešava metodom konačnih razlika, koja daje aproksimaciju rešenja za raspored temperature, primenjujući prostornu i vremensku diskretizaciju. Programska implementacija rešenja čuva temperaturu svake tačke diskretizacije u dvodimenzionoj matrici. Svaki red sadrži temperaturnu distribuciju štapa u nekom trenutku vremena. Štap je podeljen na n delova dužine h, pa stoga svaki red ima $n+1$ elemenata. Načelno, što je veće n, manja je greška aproksimacije. Vreme od 0 do T je podeljeno u m diskretnih intervala dužine k, pa stoga matrica ima $m+1$ redova, kao što je prikazano na Sl. 2.3.



Sl. 2.2: Kako vreme teče, štap se hladi. Metoda konačnih razlika omogućava izračunavanje temperature u fiksnom broju tačaka u ravnomernim vremenskim intervalima. Smanjenje prostornog i vremenskog koraka uglavnom dovodi do preciznijeg rešenja.



Sl. 2.3: Diskretizacija jednačine provođenja toplote metodom konačnih razlika

Svaka tačka $u_{i,j}$ predstavlja element matrice koji sadrži temperaturu na poziciji $i \cdot h$, u trenutku $j \cdot k$. Na krajevima štapa je temperatura uvek nula. U početnom trenutku,

temperatura u tački x je, kao što je već rečeno, $sin(\pi x)$. Algoritam ide korak po korak kroz vreme, koristi vrednosti iz trenutka j da bi izračunao vrednosti u trenutku $j+1$. Formula koja reprezentuje varijantu aproksimacije FTCS (*Forward Time Centered Space*) kao u Recktenwald [Rec04] se ovde daje bez izvođenja i glasi:

$$u_{i,j+1} = R \cdot u_{i-1,j} + (1 - 2R) \cdot u_{i,j} + R \cdot u_{i+1,j}, \tag{2.2}$$

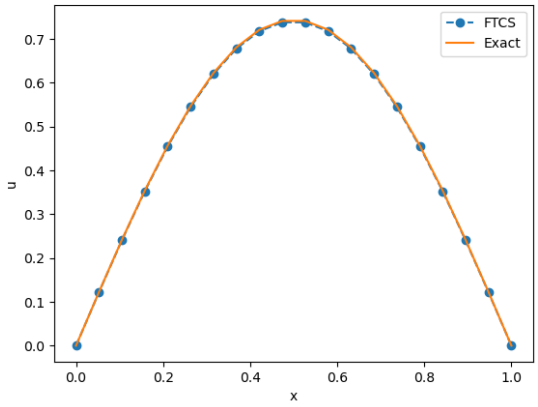
gde je

$$R = \alpha \frac{k}{h^2}.$$

Celokupna analiza različitih eksplicitnih i implicitnih metoda data je u pomenutoj referenci, a ključni deo koda u programskom jeziku Pajton implementiran je na sledeći način:

```
def heatFTCS(nt=10, nx=20, alpha=0.3, L=1, tmax=0.1):
    h = L / (nx - 1)
    k = tmax / (nt - 1)
    r = alpha * k / h**2

    x = np.linspace(0, L, nx)
    t = np.linspace(0, tmax, nt)
    U = np.zeros((nx, nt))

    # Početni uslov
    U[:, 0] = np.sin(np.pi * x / L)

    # Glavna petlja za MKR
    for m in range(1, nt):
        for i in range(1, nx-1):
            U[i, m] = r * U[i - 1, m - 1] + (1-2*r) * U[i, m-1] + r * U[i+1, m-1]

    # Egzaktno rešenje za poređenje
    ue = np.sin(np.pi * x / L) * \
        np.exp(-t[nt - 1] * alpha * (np.pi / L) * (np.pi / L))
```

Kao što detaljno objašnjava Recktenwald [Rec04], ako se MKR petlja formuliše eksplicitno kao što je to slučaj kod FTCS tehnike, mora se pažljivo izabrati vremenski i prostorni korak, kako bi numerička propagacija bila „brža" od fizičke. Rešenje koje se dobija pomoću MKR šeme se može videti na Sl. 2.4.



Sl. 2.4: Rešenje koje se dobija MKR metodom koristeći eksplicitnu FTCS tehniku u trenutku $t = 0.1s$

Ovaj problem ima i analitičko rešenje, pa je pogodan za testiranje različitih numeričkih metoda. To rešenje glasi:

$$u(x,t) = \sin\left(\frac{\pi x}{L}\right) \cdot e^{-\frac{\alpha\pi^2}{L^2}t}. \tag{2.3}$$

ili u našem slučaju, kada je $L = 1$:

$$u(x,t) = \sin(\pi x) \cdot e^{-\alpha\pi^2 t}.$$

Eksplicitne tehnike poput FTCS ne garantuju konzistentnost rešenja koju garantuju implicitne tehnike kao što je BTCS (*Backward Time Centered Space*). MKR je ustaljeni pristup koji za većinu pravilno definisanih prostornih domena radi veoma dobro. Za ovako jednostavnu postavku kao što je jednodimenziono provođenje toplote i kada su svi parametri problema poznati (ovde je to α), MKR je često optimalna metoda. Međutim, kod većine problema iz prakse to nije slučaj. Hajde da razmotrimo kako da ovaj problem rešimo koristeći NMPFZ i direktno uporedimo sa MKR.

2.2.2 Rešavanje pomoću NMPFZ

Rešenje jednačine Sl. 2.1 sa već postavljenim graničnim i početnim uslovima:

$$u(x=0,t) = u(x=1,t) = 0, \forall t$$
$$u(x,t=0) = sin(\pi x) \tag{2.4}$$

potražićemo pomoću NMPFZ pristupa. Iako je moguće da metode implementiramo direktno kao Raissi *et al.* [RPK19] koristeći okvir za duboko učenje kao što je Tensorflow[7], ipak ćemo iskoristiti pomoć biblioteka koje dodatno apstrahuju NMPFZ entitete i omogućavaju korisniku da se fokusira na problem koji rešava. Ovaj primer rešićemo koristeći biblioteku SCIANN[8] autora Haghighat and Juanes [HJ21]. Postupak rešavanja objasnićemo direktno kroz programski kod:

Listing 2.1: NMPFZ - provođenje toplote

```
1  import numpy as np
2  import sciann as sn
3  from sciann.utils.math import diff, sign, sin, sqrt, exp
4  from numpy import pi
5
6  x = sn.Variable('x')
7  t = sn.Variable('t')
8  u = sn.Functional('u', [x,t], 3*[20], 'tanh')
9  alpha = 0.3
10
11 L1 = diff(u, t) - alpha * diff(u, x, order=2)
12
13 TOLX = 0.011
14 TOLT = 0.0011
15 C1 = (1-sign(t - TOLT)) * (u - sin(pi*x))
16 C2 = (1-sign(x - (0+TOLX))) * (u)
17 C3 = (1+sign(x - (1-TOLX))) * (u)
18
19 m = sn.SciModel([x, t], [L1, C1, C2, C3], 'mse', 'Adam')
20
21 x_data, t_data = np.meshgrid(
22     np.linspace(0, 1, 101),
23     np.linspace(0, 0.1, 101)
24 )
25
26 h = m.train([x_data, t_data], 4*['zero'], learning_rate=0.002, batch_
↪size=256, epochs=500)
```

(continues on next page)

[7] https://www.tensorflow.org/
[8] https://www.sciann.com/

(nastavak sa prethodne stranice)

```
27
28  # Test
29  nx, nt = 20, 10
30  x_test, t_test = np.meshgrid(
31      np.linspace(0.01, 0.99, nx+1),
32      np.linspace(0.01, 0.1, nt+1)
33  )
34  u_pred = u.eval(m, [x_test, t_test])
```

Varijable x i t se na početku definišu na propisani način. Osnovni pojam koji se koristi u SCIANN biblioteci za apstrakciju NMPFZ je funkcional, koji je ovde označen sa u, koji kao ulaz uzima x i t, ima 3 skrivena sloja sa po 20 neurona i kao aktivaciju svih tih neurona uzima funkciju hiperboličkog tangensa. Prvi sabirak kompozitne funkcije gubitka proizilazi iz same diferencijalne jednačine (2.1). Kao što se vidi, za diferenciranje se koristi specijalni operator `diff()` iz biblioteke:

```
L1 = diff(u, t) - alpha * diff(u, x, order=2)
```

Najzanimljiviji i ne baš tako očigledan je način definisanja početnog uslova `C1` i graničnih uslova `C2` i `C3`:

```
C1 = (1-sign(t - TOLT)) * (u - sin(pi*x))
C2 = (1-sign(x - (0+TOLX))) * (u)
C3 = (1+sign(x - (1-TOLX))) * (u)
```

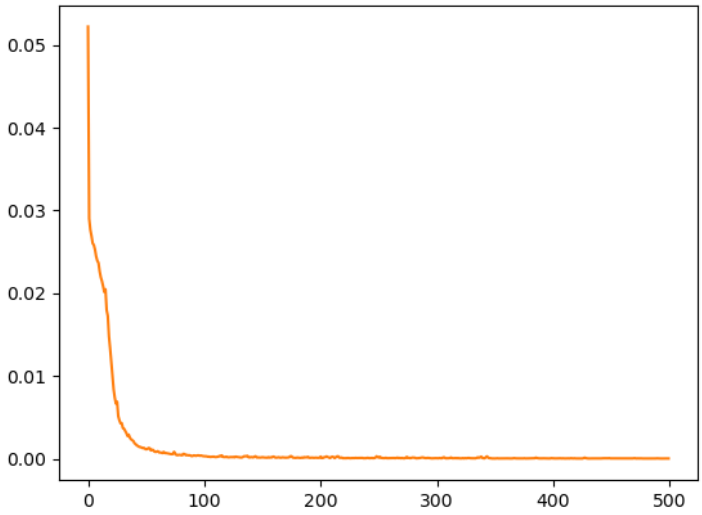
Ovde je `C1` jednak nuli u svim tačkama uzorkovanja osim za $t \leq TOLT$. Tolerancije *TOLX* i *TOLT* su postavljene tako da „hvataju" prvu/poslednju vrstu ili kolonu kolokacionih tačaka, u zavisnosti šta je potrebno. Umesto funkcije znaka `sign()`, mogu se koristiti i glatkije funkcije, kao što je hiperbolički tangens. NMPFZ model se formira pomoću `SciModel` konstruktora koji definiše i tip funkcije gubitka i algoritam optimizacije, tj. obučavanja:

```
m = sn.SciModel([x, t], [L1, C1, C2, C3], 'mse', 'Adam')
```

Obučavanje modela se pokreće metodom `train()`, pri čemu se navode sledeći parametri:

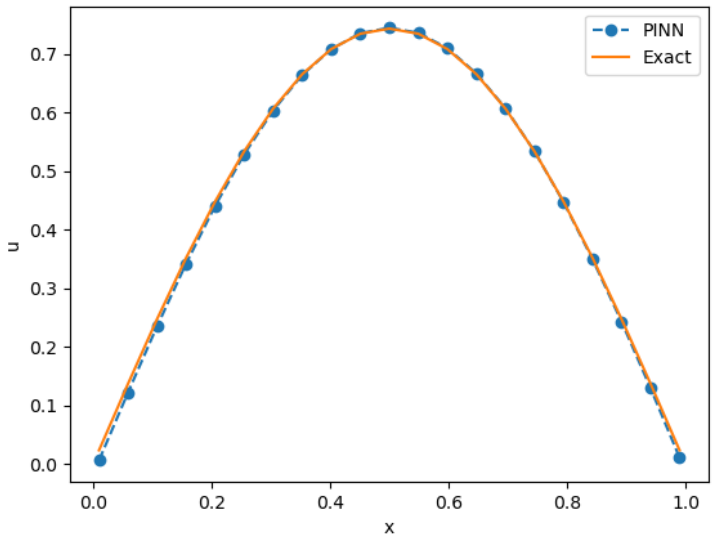
1. **Skup kolokacionih tačaka za treniranje**. Ovde je to pravilna ekvidistantna mreža tačaka po obe varijable. Horizontala je prostor, a vertikala vreme.
2. **Početne vrednosti komponenti funkcije gubitka**. Uobičajeno je da se na početku postave na nule.

3. **Stopa učenja**,
4. **Veličina batch-a**. Treba imati na umu da ako je broj tačaka domena u kojima se nameću granični uslovi značajno manji u odnosu na ukupan broj kolokacionih tačaka, parametar optimizacije `batch_size` treba da bude podešen na veliki broj koji garantuje doslednu optimizaciju mini *batch*-a. U suprotnom, može da se desi da neki mini *batch*-evi ne dobiju nijednu kolokacionu tačku koja pripada graničnim uslovima i stoga ne generišu tačan gradijent za ažuriranje metodom gradijentnog spusta.
5. **Broj epoha**.



Sl. 2.5: Istorija obuke jednodimenzionog modela provođenja toplote.

Tok obuke možemo da ispratimo kroz standardne *Tensorflow* objekte, kao što je `h.history['loss']`, kao što se vidi na Sl. 2.5. Pošto se završi obuka NMPFZ-a, možemo formirati testni skup tačaka slično kao što smo to učinili i sa kolokacionim tačkama i proveriti rezultate predikcije pozivom metode `eval()` na objektu istreniranog modela. Rezultat polja temperature duž štapa u trenutku $t = 0,1$ i njegovo poređenje sa analitičkim rešenjem vidi se na Sl. 2.6.



Sl. 2.6: Polje temperature duž štapa u trenutku $t = 0,1$ dobijeno metodom NMPFZ.

Čisto praktično gledano, **NMPFZ rešenje jednostavnog direktnog problema kao što je ovaj i ne pruža nikakve posebne prednosti u odnosu na klasičnu MKR metodu**. Prvo, rešavanje duže traje i zahteva upošljavanje više računarskih resursa i zavisnosti u vidu dodatnih biblioteka za tenzorski račun. Dalje, specifikacija početnih i graničnih uslova kod NMPFZ ima svoje specifičnosti. Treće, neophodno je metodom probe i greške podesiti hiper-parametre modela, kao što su: broj skrivenih slojeva, broj neurona po sloju, aktivaciona funkcija, brzina učenja (*Learning Rate*) itd. Od izbora hiper-parametara konvergencija rešenja može značajno da zavisi.

Sa druge strane, za razliku od MKR i MKE (*Metoda Konačnih Elemenata*), NMPFZ nam dozvoljava da problem definišemo čistim diferencijalnim jednačinama i proizvoljnim graničnim uslovima (Dirihleovi, Nojmanovi, periodični, skup tačaka). **Nema potrebe za specificiranjem algebarske veze između čvorova** (tj. kolokacionih tačaka u NMPFZ) i rešavanjem tako postavljenog sistema jednačina. Zahvaljujući ovoj činjenici, bilo koja nova fizika u vidu novog graničnog uslova ili promena u samoj diferencijalnoj jednačini može da se izvede veoma lako, omogućavajući brzu proveru hipoteza i izradu prototipova.

Drugo, dok sve klasične metode proračun moraju da izvedu kroz vremenske korake (*time stepping*), **NMPFZ omogućava brzu inferenciju** na već obučenoj mreži za bilo koji vremenski trenutak t postavljen na ulazu mreže. Za neke primene u realnom vremenu

gde je brzina od ključnog značaja, ovo može da bude presudno.

Treće, NMPFZ metodološki ne razlikuje **direktne probleme** (u kojima se rešava poznata diferencijalna jednačina) od **inverznih problema**, kod kojih su neki od parametara nepoznati, ali postoje dodatni uslovi iz kojih se nepoznati parametri mogu dobiti procesom treninga. U narednoj temi *Jednodimenzioni inverzni problem* (stranica 22) demonstriraćemo jedan takav problem.

2.3 Jednodimenzioni inverzni problem

Postavka inverznog problema je potpuno ista kao i u prethodnom poglavlju *Jednodimenzioni direktni problem* (stranica 14), tj. opisana je Sl. 2.1, jednačinom (2.1) i graničnim uslovima (2.4). Međutim, ovoga puta nam parametar problema α na početku nije poznat i pokušaćemo da ga dobijemo uz pomoć metoda obučavanja propagacijom unazad. Naravno, čim smo uveli novu nepoznatu, moramo da uvedemo i novi granični uslov. Recimo, možemo da postavimo da je u jednoj tački u nekom vremenskom trenutku, temperatura u odgovarala nekoj numeričkoj vrednosti koja se poklapa sa analitičkim rešenjem (2.3). Recimo, postavimo temperaturu na sredini štapa u $x = 0,5$ u trenutku $t = 0,05$ na:

$$u(x = 0,5, t = 0,05) = 0,8623931,$$

i pokušajmo da rešimo problem postavljajući granične uslove na sledeći način:

Listing 2.2: Pronalaženje nepoznatog parametra α

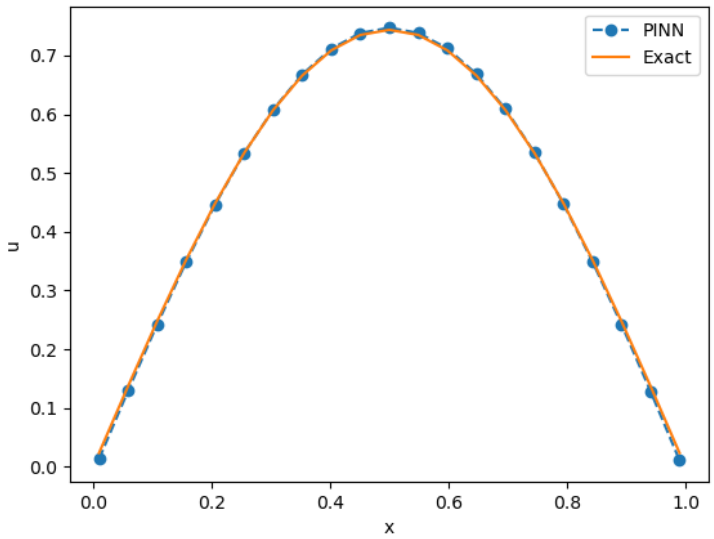
```
x = sn.Variable('x')
t = sn.Variable('t')
u = sn.Functional('u', [x,t], 3*[20], 'tanh')
alpha = sn.Parameter(0.5, inputs=[x,t], name="alpha")

L1 = diff(u, t) - alpha * diff(u, x, order=2)

TOL = 0.011
TOLT= 0.0011
C1 = (1-sign(t - TOLT)) * (u - sin(pi*x))
C2 = (1-sign(x - (0+TOL))) * (u)
C3 = (1+sign(x - (1-TOL))) * (u)
C4 = (1 + sign(t-0.049)) * (1 - sign(t-0.051)) * (1 + sign(x-0.49)) *␣
↪(1 - sign(x-0.51)) * (u-0.8623931)

m = sn.SciModel([x, t], [L1, C1, C2, C3, C4], 'mse', 'Adam')

```

(continues on next page)

(nastavak sa prethodne stranice)

```
17 x_data, t_data = np.meshgrid(
18     np.linspace(0, 1, 101),
19     np.linspace(0, 0.1, 101)
20 )
21
22 h = m.train([x_data, t_data], 5*['zero'], learning_rate=0.002, batch_
    ↪size=512, epochs=1200,
23     adaptive_weights={'method':'NTK', 'freq':100})
24
25 # Test
26 nx, nt = 20, 10
27 x_test, t_test = np.meshgrid(
28     np.linspace(0.01, 0.99, nx+1),
29     np.linspace(0.01, 0.1, nt+1)
30 )
31 u_pred = u.eval(m, [x_test, t_test])
32
33 print(alpha.value)
```

Očigledno je da je kod gotovo isti kao prethodni u kome se rešava direktni problem, jer je i metodologija za rešavanje direktnih i inverznih problema kod NMPFZ identična. Jedina razlika je u postavci. Linija 4 postavlja α kao nepoznati parametar i daje mu početnu vrednost. U liniji 13 se postavlja dodatni granični uslov u tački `u(x=0.4,t=0.05)`, koji će postati još jedna komponenta kompozitne funkcije gubitka koja se formira u liniji 15. Vrednost nepoznatog parametra se štampa u poslednjoj liniji i u našem testu iznosi oko 0,308, što je dovoljno blisko realnoj vrednosti od 0,3. Potvrda zadovoljavajućeg rešenja inverznog problema prikazana je i grafički na Sl. 2.7.



Sl. 2.7: Polje temperature duž štapa u trenutku $t = 0,1$ dobijeno rešavanjem inverznog problema

2.4 Stefanov problem u modelima topljenja

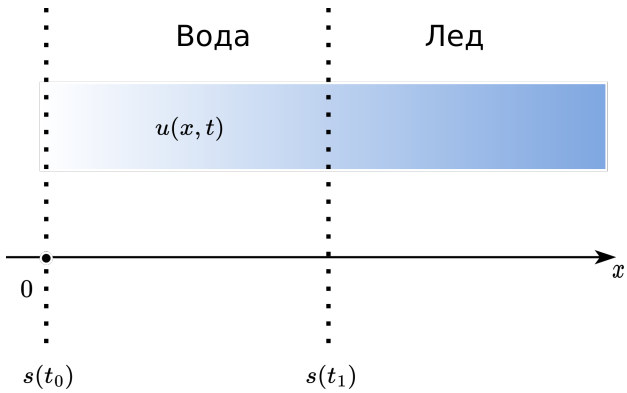
Nakon što smo uspešno rešili i direktni i inverzni problem provođenja toplote koji je opisan jednostavnom paraboličnom parcijalnom diferencijalnom jednačinom, pozabavićemo se malo komplikovanijom postavkom sličnog problema, pre svega imajući u vidu granične uslove. Ovde će se oni dinamički menjati u zavisnosti od gradijenta temperature. Pokazaćemo da je zapravo ovako postavljen problem lakše i prirodnije rešavati pomoću NMPFZ-a nego pomoću klasičnih numeričkih metoda baziranih na mreži integracionih tačaka (MKE, MKR).

Stefanovi problemi faznih promena imaju primenu u raznim oblastima nauke i inženjerstva, kada god se faza posmatrane supstance menja između tečnog, čvrstog ili gasovitog stanja. Pretpostavlja se da materijal prolazi kroz faznu promenu sa kretanjem granice, čiji je položaj nepoznat i mora se odrediti kao deo same numeričke analize. Pošto problemi sa pomeranjem granica zahtevaju rešavanje toplotne jednačine u nepoznatoj oblasti koja se takođe mora odrediti kao deo rešenja, oni su inherentno nelinearni.

Problem jednodimenzionalne promene faze mogao bi se demonstrirati pomoću polube-

skonačnog čvrstog tela, kao što je tanak blok leda koji zauzima $0 \leq x < \infty$, na temperaturi očvršćavanja. Na fiksnoj granici tankog bloka leda $x = 0$, mogu da deluju različite vrste fluksa. U ovom primeru koristimo isti granični uslov kao Ivanovic *et al.* [ISS17], Savovic and Caldwell [SC09], tako se temperatura pri $x = 0$ povećava eksponencijalno sa vremenom. Takođe, propisujemo da se čitava čvrsta faza nalazi na temperaturi topljenja. Stoga svodimo problem na određivanje raspodela temperature u tečnoj fazi u vreme t_0, gde je $x < s(t_0)$, kao i položaja granice između faza $s(t_0)$.

U nekom kasnijem vremenskom trenutku $t_1 > t_0$, pokretna granica $s(t)$ kreće se udesno, zauzimajući poziciju $s(t_1) > s(t_0) = s_0$, kao što je prikazano na Sl. 2.8. Deo tankog bloka leda od pozicije $s(t_0)$ do pozicije $s(t_1)$, otopio se tokom vremenskog intervala (t_0, t_1).



Sl. 2.8: Stefanov problem u jednoj dimenziji. $s(t)$ označava pokretnu granicu, a $u(x,t)$ temperaturu tečne faze (za $x < s$).

Distribucija temperature $u(x,t)$ u regionu u kome vlada tečno stanje $0 \leq x \leq s(t)$ data je toplotnom jednačinom:

$$\frac{\partial u}{\partial t} = \alpha \cdot \frac{\partial^2 u}{\partial x^2},$$

koja može da se napiše na sledeći način:

$$\frac{\partial u}{\partial t} - \alpha \cdot \frac{\partial^2 u}{\partial x^2} = 0, \tag{2.5}$$

pod sledećim graničnim uslovima:

$$\begin{aligned} u(x,t) &= e^{\alpha t}, \quad x = 0, \quad t > 0 \\ u(x,t) &= 1, \quad x = s(t), \quad t > 0. \end{aligned} \tag{2.6}$$

Ovde α, kao i u prethodnim primerima označava fizički parametar koji kombinuje toplotnu provodnost, gustinu i specifičnu toplotu. Pozicija pokretne granice data je jednačinom koja je poznata kao Stefanov uslov:

$$\frac{1}{\alpha} \cdot \frac{ds}{dt} = -\frac{\partial u}{\partial x}, \quad x = s(t), \quad t > 0. \tag{2.7}$$

U opštem slučaju, početni uslov za položaj granice faza dat je sa:

$$s(0) = 0. \tag{2.8}$$

Za ovako postavljen problem poznato je i analitičko rešenje, i to:

$$u(x,t) = e^{\alpha t - x}$$
$$s(t) = \alpha t.$$

Rešavanje ovog problema NMPFZ pristupom podrazumeva konstrukciju dve neuronske mreže. Prva će aproksimirati distribuciju temperatura $u(x,t)$ dok će druga aproksimirati položaj slobodne granice između faza $s(t)$. Aproksimativna rešenja biće automatski diferencirana u odnosu na ulazne varijable od kojih zavise, za vrednosti definisane skupom kolokacionih tačaka iz domena $[0,T] \times \mathcal{D}$, gde je $\mathcal{D} \subset \mathbb{R}^d$ ograničeni domen, a T označava konačno vreme simulacije. Funkcija gubitka sastoji se iz komponenti izvedenih iz (2.5), (2.6) i (2.8), koristeći aproksimacije za u i s u kolokacionim tačkama, koje pokrivaju kako unutrašnjost domena, tako i domene u kojima važe početni i granični uslovi.

2.4.1 Konstrukcija funkcije gubitka

Kao što je već rečeno, prva mreža aproksimira funkciju temperature $u(x,t)$, a druga mreža aproksimira položaj slobodne granice između faza $s(t)$. Funkcija gubitka sastoji se iz razlike u i s i njihovih aproksimacija $\widehat{u}$ i $\widehat{s}$ koje daje NMPFZ i koji predstavljaju reziduume koje daje glavna diferencijalna jednačina, početni i granični uslovi. Dakle, ukupan gubitak $\mathcal{L}$ određen je sumom reziduuma:

$$\mathcal{L} = \mathcal{L}_r + \mathcal{L}_0 + \mathcal{L}_{b_1} + \mathcal{L}_{b_2} + \mathcal{L}_{b_3}, \tag{2.9}$$

gde su:

$$\begin{aligned}
\mathcal{L}_r &= \frac{1}{N_r}\sum_{i=1}^{N_r}\left|\frac{\partial\widehat{u}(x,t)}{\partial t} - \alpha\frac{\partial^2\widehat{u}(x,t)}{\partial x^2}\right|^2, \\
\mathcal{L}_0 &= \frac{1}{N_0}\sum_{i=1}^{N_0}|\widehat{s}(0) - s(0)|^2, \\
\mathcal{L}_{b_1} &= \frac{1}{N_{b_1}}\sum_{i=1}^{N_{b_1}}\left|\frac{1}{a}\frac{\partial\widehat{s}(t)}{\partial t} + \frac{\partial\widehat{u}}{\partial\widehat{s}(t)}\right|^2, \\
\mathcal{L}_{b_2} &= \frac{1}{N_{b_2}}\sum_{i=1}^{N_{b_2}}|\widehat{u}(0,t) - u(0,t)|^2, \\
\mathcal{L}_{b_3} &= \frac{1}{N_{b_3}}\sum_{i=1}^{N_{b_3}}|\widehat{u}\left(\widehat{s}(t),t\right) - u\left(s(t),t\right)|^2.
\end{aligned} \tag{2.10}$$

Prvi član $\mathcal{L}_r$ penalizuje po glavnoj diferencijalnoj jednačini (2.5), gde je N_r veličina *batch*-a kolokacionih tačaka koje se slučajno uzorkuju iz domena prostor-vremenskih koordinata koje uzimaju vrednosti $0 \leq x \leq 1$ i $0s \leq t \leq 0,5s$, respektivno. $\widehat{u}(x,t)$ je aproksimativna neuronska mreža temperaturskog polja $u(x,t)$. Drugi član $\mathcal{L}_0$ određuje ispunjenost graničnog uslova (2.8). Ispunjenost Stefanovog graničnog uslova (2.7) dat je reziduumom $\mathcal{L}_{b_1}$, gde $\widehat{s}(t)$ označava NMPFZ aproksimaciju položaja pokretne granice. Poslednja dva člana $\mathcal{L}_{b_2}$ i $\mathcal{L}_{b_3}$ određuju reziduale graničnih uslova (2.6), gde N_0, N_{b_1}, N_{b_2}, i N_{b_3} označavaju broj kolokacionih tačaka u kojima važe početni i granični uslovi.

2.4.2 Implementacija

Rešenje koje koristi funkcionalnost već poznate biblioteke SCIANN dato je u sledećem listingu:

Listing 2.3: Rešenje Stefanovog problema u 1D korišćenjem SCIANN biblioteke

```
alpha = 1.0

# Pocetni uslovi
t0 = 0.1
s0 = alpha * t0

# Varijable
```

(continues on next page)

(nastavak sa prethodne stranice)

```
x = sn.Variable('x')
t = sn.Variable('t')
u = sn.Functional (["u"], [x, t], 3*[30] , 'tanh')
s = sn.Functional (["s"], [t], 3*[30] , 'tanh')

# Glavna dif. jednacina
L1 =  diff(u, t) - alpha * diff(u, x, order=2)

TOLX=0.004
TOLT=0.002

# Stefanov uslov
C1 = (1/alpha*diff(s, t) + diff(u,x)) * (1 + sign(x - (s-TOLX))) * (1-
↪sign(x-s))
# Pocetno s u trenutku t=t0
C2 = ( s - s0 ) * (1-sign(t - (t0+TOLT)))
# Granicni uslov za u kada je x=0
C3 = ( u - exp(alpha*t) ) * (1-sign(x - (0 +TOLX)))
# Temperatura na granici izmedju faza je 1
C4 = (u-1) * (1-sign(x - (s+TOLX))) * (1+sign(x-s))

x_data, t_data = [], []

# Trening skup
x_train, t_train = np.meshgrid(
    np.linspace(0, 1, 300),
    np.linspace(t0, 0.5, 300)
)

x_data, t_data = np.array(x_train), np.array(t_train)

m = sn.SciModel([x, t], [L1,C1,C2,C3,C4], 'mse', 'Adam')
h = m.train([x_data, t_data], 5*['zero'], learning_rate=0.002, batch_
↪size=1024, epochs=200, adaptive_weights={'method':'NTK', 'freq':20})

# Test
x_test, t_test = np.meshgrid(
    np.linspace(0, 1, 30),
    np.linspace(0.01, 0.5, 30)
)
u_pred = u.eval(m, [x_test, t_test])
s_pred = s.eval(m, [x_test, t_test])

s=[]
for e in s_pred:
    s.append(e[0])
```

Na početku (linije 1-11) postavljamo konstante i varijable. Primetimo da modelovanje kreće od vremenskog trenutka $t_0 = 0,1s$. Prva zagrada u izrazu za `C1`

```
C1 = (1/alpha*diff(s, t) + diff(u,x)) * (1 + sign(x - (s-TOLX))) * (1-
↪sign(x-s))
```

predstavlja sam Stefanov uslov. Kao što je uobičajeno za postavljanje graničnih uslova, druga zagrada postavlja pravilo gde taj granični uslov važi. Za `C1` je taj izraz malo komplikovaniji zbog tolerancije, ali zapravo predstavlja uslov da je $x \approx s$. Uslov

```
C2 = ( s - s0 ) * (1-sign(t - (t0+TOLT)))
```

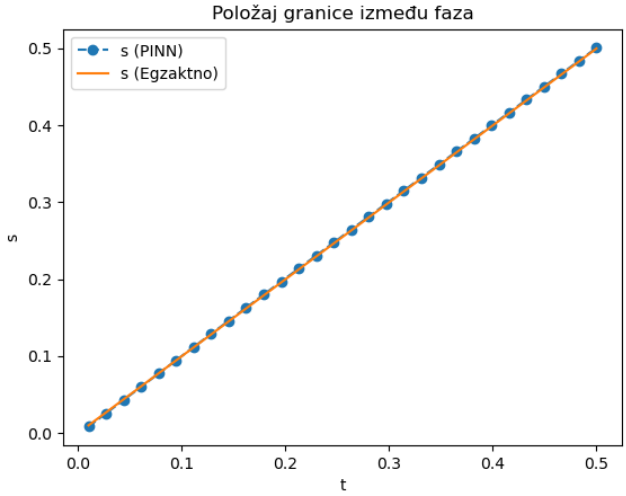
definiše položaj granice u početnom trenutku. Dalje, uslov

```
C3 = ( u - exp(alpha*t) ) * (1-sign(x - (0 +TOLX)))
```

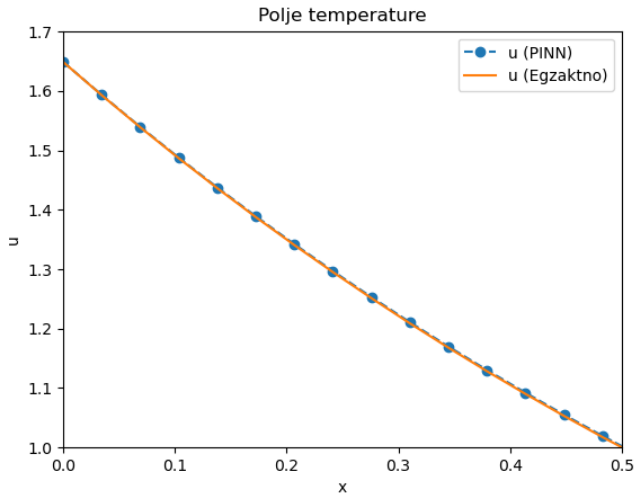
postavlja dinamički uslov promene temperature u tački $x = 0$ prema jednačini (2.6). Konačno, posledlji uslov

```
C4 = (u-1) * (1-sign(x - (s+TOLX))) * (1+sign(x-s))
```

u istom skupu kolokacionih tačaka kao što je to bio slučaj sa uslovom `C1`, tj. na granici između faza $x \approx s$ postavlja vrednost temperature na 1. Ostatak koda je manje-više standardno treniranje, formiranje testnog skupa i ekstrakcija podataka o kretanju granice `s` kroz vreme. Kretanje granice između faza kroz vreme može se videti na Sl. 2.9, dok se polje temperature $u(x, t = 0,5)$ može videti na Sl. 2.10.



Sl. 2.9: Kretanje granice između faza $s(t)$ tokom vremena.



Sl. 2.10: Polje temperatura $u(x,t)$ u trenutku $t = 0,5$

Sa dijagrama je očigledno da se NMPFZ (*PINN*) rešenja u zadovoljavajućoj meri slažu sa analitičkim rešenjem za posmatranu pojavu. U skladu sa prethodnom opštom diskusijom o upotrebnoj vrednosti NMPFZ pristupa, kod direktnih problema kao što je ovaj,

dodatna vrednost u odnosu na klasične numeričke metode može se naći u jednostavnijoj formulaciji. Međutim, tek kod inverznih problema, kada su i neki od parametara nedovoljno poznati, pristup učenja propagacijom unazad pokazuje svoju pravu snagu. Sledeći odeljak uvodi nepoznati parametar u ovaj isti problem.

2.4.3 Inverzni 1-D Stefanov problem

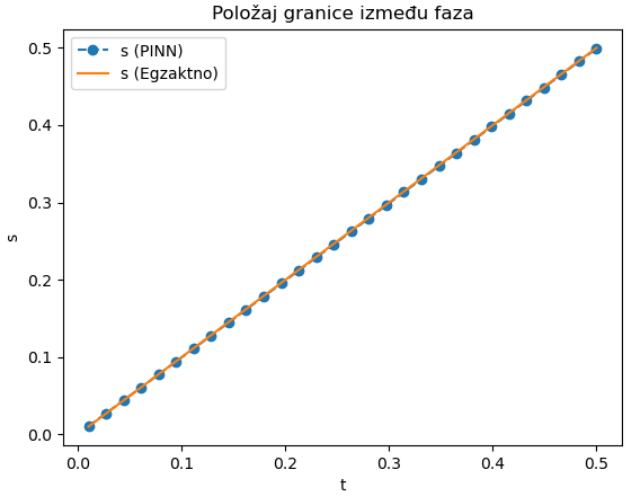
Pretpostavim identičnu postavku Stefanovog problema, osim što sada nije poznata vrednost materijalnog parametra α, pa time ni potpuni oblik diferencijalne jednačine, ali je zato poznato da je, na primer, u trenutku $t = 0,2$ granica između faza uočena na koordinati $x = 0,2$. Nepoznati parametar uvešćemo formulacijom:

```
# Nepoznati parametar
alpha = sn.Parameter(2.5, inputs=[x,t])
```

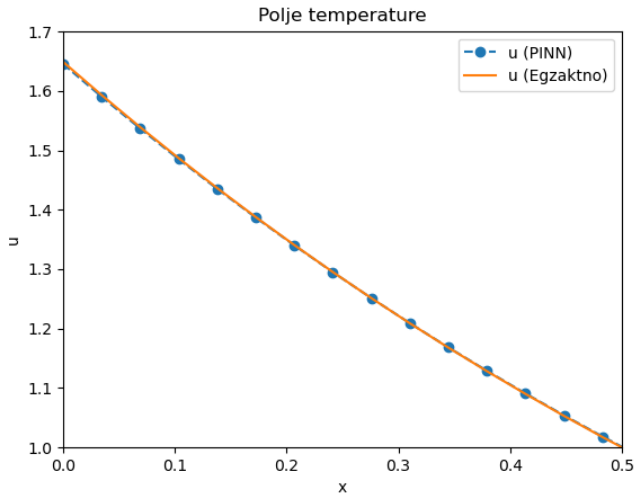
Parametru je data početna vrednost od 2,5 i postavljena zavisnost od obe ulazne varijable x i t. Dodatni granični uslov `C5` dat je standardno:

```
# Dodatni uslov u tacki s(t=0.2)=0.2
C5 = (1-sign(t - (0.2+TOLT))) * (1+sign(t-0.2)) * (s-0.2)
```

Iako je ovog puta potrebno nešto više epoha u procesu treninga, jer je početna vrednost nepoznatog parametra α (2,5) prilično daleko od realne vrednosti (1), algoritam koji koristi brzinu učenja od 0,001 u stotinak epoha dolazi do vrednosti $\alpha = 0,99$ i zaista zadovoljavajućeg poklapanja sa analitičkim rešenjem, kao što se vidi na Sl. 2.11 za kretanje granice i za polje temperature u $u(x, t = 0,5)$ na Sl. 2.12.



Sl. 2.11: Kretanje granice između faza $s(t)$ tokom vremena za inverzni problem

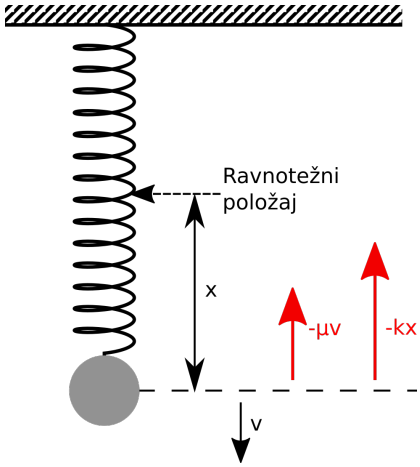


Sl. 2.12: Polje temperatura $u(x,t)$ u trenutku $t = 0,5$ za inverzni problem

3 Mehaničke oscilacije

3.1 Oscilator

Jedno od najjednostavnijih periodičnih kretanja u mehanici je oscilatorno kretanje (oscilovanje), gde se telo kreće po istoj putanji, ali menja smer kretanja. Ako bismo razmatrali oscilatorno kretanje u idealnim uslovima, bez trenja i otpora sredine, ono bi trajalo beskonačno dugo. Pri takvim slobodnim, **neprigušenim oscilacijama**, oscilator ne gubi energiju, a amplituda se ne menja u toku vremena. Međutim, u realnim uslovima, mora da se uzme u obzir uticaj okoline na kretanje tela. Oscilovanje usporava sa vremenom, smanjuju se amplitude, jer se ukupna mehanička energija troši na savladavanje otpora sredine. To su tzv. **prigušene oscilacije**, kada oscilator gubi energiju i amplituda se smanjuje u toku vremena. Treći slučaj bi bio kada gubitak energije oscilatora može da se nadoknadi delovanjem spoljašnje periodične sile. Amplituda oscilovanja će pri ovakvom načinu kretanja ostati konstantna, ukoliko se pri svakom ciklusu kretanja u sistem doda energija jednaka onoj energiji koju je sistem izgubio. Ovakvo kretanje se naziva **prinudno oscilovanje**.



Sl. 3.1: Postavka eksperimenta sa oprugom i tegom. Na teg deluju sila elastičnosti opruge i sila trenja.

Problem koji rešavamo predstavlja modelovanje oscilovanja tega koji visi na opruzi, kao što je prikazano na Sl. 3.1. Parametri koji utiču na kretanje su masa tega, koeficijent trenja i koeficijent elastičnosti opruge. Sile koje utiču na kretanje oscilatora su:

- **Povratna sila opruge** $F_f = k \cdot x$ vuče teg ka tački mirovanja i direktno je proporcionalna otklonu klatna. Njen smer je suprotan od smera otklona. Parametar k predstavlja konstantu opruge.
- **Sila trenja** $F_r = -\mu \cdot \dot{x}$ je proporcionalna brzini tega, dok je smer uvek suprotan smeru kretanja tega. Parametar μ predstavlja koeficijent trenja.

Dakle, sila inercije je suprotstavljena dvema silama:

$$m\ddot{x} = -\mu\dot{x} - kx$$

ili transformisano:

$$\ddot{x} + \frac{\mu}{m}\dot{x} + \frac{k}{m}x = 0 \tag{3.1}$$

Ova jednačina je linearna homogena diferencijalna jednačina drugog reda sa konstantnim koeficijentima. Za analitičko rešavanje diferencijalne jednačine ovog tipa koristi se eksponencijalna funkcija oblika:

$$x(t) = Ce^{\lambda t}$$

Prvi i drugi izvod ove jednačine glase:

$$\dot{x}(t) = \lambda C e^{\lambda t}, \qquad \ddot{x}(t) = \lambda^2 C e^{\lambda t} \tag{3.2}$$

Zamenom jednačina (3.2) u diferencijalnu jednačinu (3.1) i skraćivanjem dobijamo:

$$\lambda^2 + \frac{\mu}{m}\lambda + \frac{k}{m} = 0 \tag{3.3}$$

Ova jednačina se naziva **karakterističnom jednačinom**. Kako je u pitanju kvadratna jednačina, razmatramo dva rešenja:

$$\lambda_{1,2} = -\frac{\mu}{2m} \pm \sqrt{\left(\frac{\mu}{2m}\right)^2 - \frac{k}{m}}.$$

Da bismo dodatno uprostili izraz, uvodimo nove konstante δ i ω_0:

$$\delta = \frac{\mu}{2m}, \qquad \omega_0 = \sqrt{\frac{k}{m}},$$

pa se rešenja karakteristične jednačine mogu izraziti u obliku:

$$\lambda_{1,2} = -\delta \pm \sqrt{\delta^2 - \omega_0^2}. \tag{3.4}$$

U zavisnosti od izbora konstanti δ i ω_0, diskriminanta može biti: veća od nule, manja od nule ili jednaka nuli. Stoga λ_1 i λ_2 mogu biti:

- dva realna različita rešenja,
- dva konjugovano kompleksna rešenja i
- dva jednaka realna rešenja.

Svaki od ovih slučajeva zahteva drugačiji pristup analitičkom rešavanju. Uopšteno rešenje homogene jednačine ima oblik:

$$x(t) = C_1 \cdot x_1(t) + C_2 \cdot x_2(t), \tag{3.5}$$

gde funkcije $x_1(t)$ i $x_2(t)$ zavise od vrednosti determinante u jednačini (3.4). Sada ćemo razmotriti sve navedene slučajeve.

3.1.1 Prepriguseni slučaj

Ukoliko je $\delta > \omega_0$, onda dominira sila trenja. Stoga je diskriminanta u jednačini (3.4) pozitivna i postoje dva različita realna rešenja $\lambda_1 \neq \lambda_2$:

$$x_1(t) = C_1 e^{\lambda_1 t}, \qquad x_2(t) = C_2 e^{\lambda_2 t}.$$

Zamenom u jednačinu (3.5) dobijamo opšte rešenje diferencijalne jednačine:

$$x(t) = C_1 e^{\left(-\delta + \sqrt{\delta^2 - \omega_0^2}\right)t} + C_2 e^{\left(-\delta - \sqrt{\delta^2 - \omega_0^2}\right)t}.$$

Uslov koji smo naveli za prepriguseni slučaj nalaže da je

$$\delta > \sqrt{\delta^2 - \omega_0^2},$$

pa je stoga rešenje zbir dveju eksponencijalnih opadajućih funkcija. Kako bismo dalje uprostili izraz, zamenićemo koren novom konstantom:

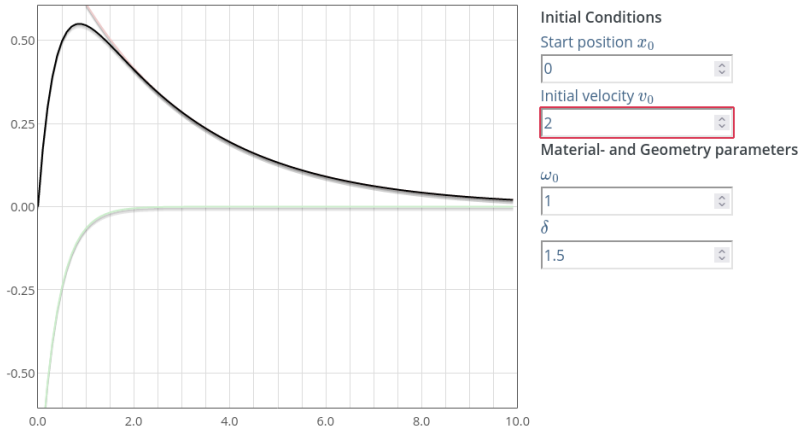
$$\alpha = \sqrt{\delta^2 - \omega_0^2},$$

pa konačno rešenje za prepriguseni slučaj glasi:

$$x(t) = e^{-\delta t}\left(C_1 e^{\alpha t} + C_2 e^{-\alpha t}\right).$$

Integracione konstante C_1 i C_2 možemo dobiti iz početnih uslova, tj. početnog položaja i početne brzine tega:

$$x(0) = x_0, \quad \dot{x}(0) = v_0.$$



Sl. 3.2: Promena položaja tega u toku vremena za prepriguseni slučaj oscilovanja. Dobijeno pomoću simulatora[Stranica 37, 9].

Na Sl. 3.2 možemo videti grafički prikaz opšteg analitičkog rešenja za preprigušeni slučaj oscilatora. Ovaj dijagram prikazuje kretanje tega tokom vremena. Crvena linija označava komponentu rešenja $C_1 e^{\lambda_1 t}$, dok zelena označava drugu komponentu rešenja $C_2 e^{\lambda_1 t}$. Crna linija je zbir ova dva parcijalna rešenja i predstavlja ukupno rešenje diferencijalne jednačine preprigušenog slučaja za zadate početne uslove.

3.1.2 Kritično-prigušeni slučaj

Ovaj slučaj se dešava kada je $\delta = \omega_0$. U ovom slučaju jednačina (3.4) ima samo jedno rešenje

$$\lambda = \lambda_1 = \lambda_2 = -\delta$$

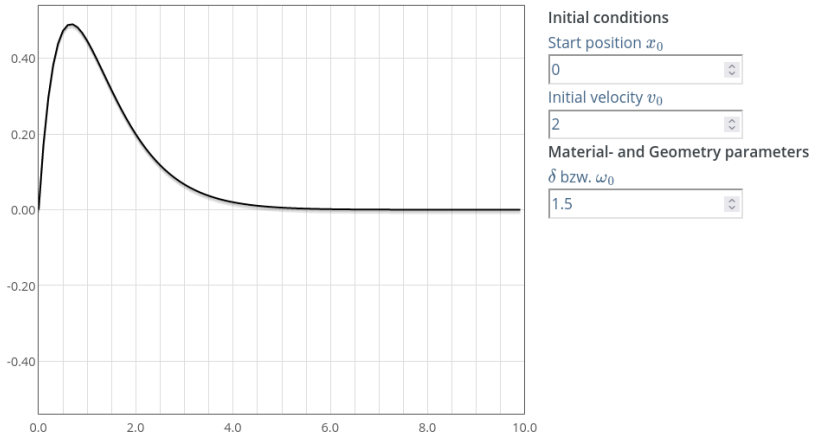
Dve komponente rešenja diferencijalne jednačine su onda:

$$x_1(t) = C_1 e^{\lambda t}, \qquad x_2(t) = t C_2 e^{\lambda t}.$$

Zamenom ovih izraza u opšte rešenje (3.5) dobijamo:

$$x(t) = e^{-\delta t}(C_1 + tC_2)$$

Integracione konstante C_1 i C_2 možemo dobiti iz početnih uslova, tj. početnog položaja i početne brzine tega.



Sl. 3.3: Promena položaja tega u toku vremena za kritično prigušeni slučaj oscilovanja. Dobijeno pomoću simulatora[Stranica 38, 10].

[9] https://beltoforion.de/en/harmonic_oscillator/

Na Sl. 3.3 vidimo grafički prikaz opšteg rešenja kritično prigušenog slučaja. Na dijagramu opažamo kretanje klatna tokom vremena za zadate početne uslove.

3.1.3 Podprigušeni slučaj

Podprigušeni slučaj nastupa kada je $\delta < \omega_0$, tj. diskriminanta jednačine (3.4) je negativna. Stoga su λ_1 i λ_2 kompleksni brojevi. Eksponencijalni izraz

$$x(t) = Ce^{\lambda t}$$

se ponovo koristi za dobijanje komponenti rešenja ove diferencijalne jednačine:

$$x_1(t) = C_1 e^{\lambda_1 t}, \qquad x_2(t) = C_2 e^{\lambda_2 t}.$$

Zamenom u izraz (3.5) i zamenom λ dobijamo:

$$x(t) = e^{-\delta t}\left(C_1 e^{\sqrt{\delta^2-\omega_0^2}t} + C_2 e^{-\sqrt{\delta^2-\omega_0^2}t}\right). \tag{3.6}$$

Sada radimo sa kompleksnim rešenjima jer su vrednosti negativne. Stoga konstante C_1 i C_2 imaju kompleksne vrednosti. Za rad sa kompleksnim vrednostima koristimo Ojlerovu formulu:

$$e^{i\phi} = \cos\phi + i\sin\phi$$

Korisno bi bilo da izmenimo jednačinu tako što bismo razdvojili imaginarne delove:

$$\sqrt{\delta^2-\omega_0^2} = \sqrt{-1\cdot(\omega_0^2-\delta^2)} = i\sqrt{\omega_0^2-\delta^2}.$$

Dobijamo deo koji se sastoji od imaginarne vrednosti i pomnoženom korenom realnih vrednosti. Da bismo uprostili dalja izračunavanja, zamenjujemo koren novom konstantom

$$\omega = \sqrt{\omega_0^2-\delta^2}$$

Ova konstanta predstavlja **prirodnu frekvenciju harmonijskog oscilatora**. Tada jednačina (3.6) može da se transformiše u:

$$x(t) = e^{-\delta t}\left(C_1 e^{i\omega t} + C_2 e^{-i\omega t}\right).$$

[10] https://beltoforion.de/en/harmonic_oscillator/

Sa fizičke strane, interesuju nas samo realne vrednosti. Da bismo ih pronašli, neophodno je da razdvojimo imaginarni i realni deo. Kao što je već pomenuto, konstante C_1 i C_2 su konstante sa kompleksnim vrednostima, a njihov polarni oblik je:

$$C_1 = \hat{C}_1 e^{i\phi_1}, \qquad C_2 = \hat{C}_2 e^{i\phi_2}$$

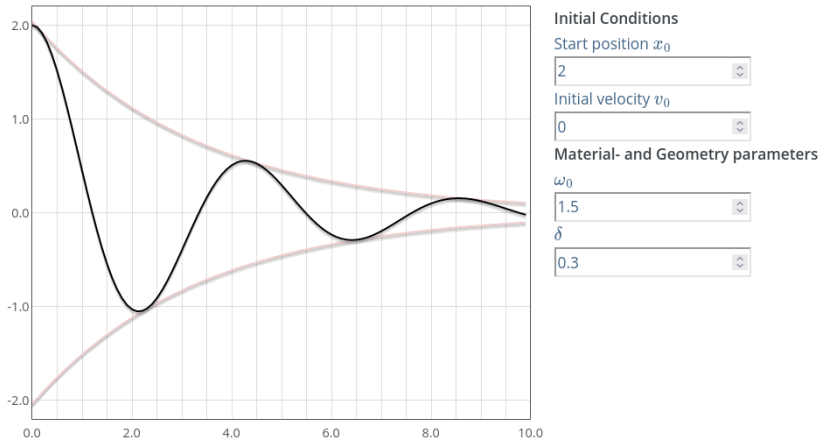
Kako analitičko rešavanje ovog problema nije u fokusu ovog materijala, nećemo do kraja analitički izvoditi izraz, već samo navesti krajnje rešenje za podprigušeni slučaj, nakon odabira realnih rešenja i transformacije i odabira odgovarajućih konstanti:

$$x(t) = e^{-\delta t}(2A\cos(\phi + \omega t))$$

A je amplituda, a ϕ je fazni pomeraj. Konstante se mogu dobiti iz početnih uslova, kao što su:

$$x(0) = x_0, \quad \dot{x}(0) = v_0.$$

Ovim smo izveli sva potrebna analitička rešenja sa kojima ćemo porediti rešenje dobijeno pomoću NMPFZ pristupa.



Sl. 3.4: Promena položaja tega u toku vremena za podprigušeni slučaj oscilovanja. Dobijeno pomoću simulatora[11].

[11] https://beltoforion.de/en/harmonic_oscillator/

3.2 Implementacija

Da bismo realizovali predloženi model prigušenih oscilacija u jednoj dimenziji opisan običnom diferencijalnom jednačinom u sekciji *Oscilator* (stranica 33), umesto biblioteke SCIANN koristićemo nešto noviju biblioteku DeepXDE Lu *et al.* [LMMK21]. Ideja je da ovim materijalom pokrijemo sve značajne softverske okvire koji podržavaju metodologiju NMPFZ u trenutku pisanja (2022-2023). Kako autori kažu, DeepXDE biblioteka je dizajnirana da služi i kao obrazovno sredstvo koje će se koristiti u visokom školstvu i kao istraživački alat za rešavanje problema u kompjuterskim naukama i inženjerstvu. Konkretno, DeepXDE može da rešava probleme za unapred date početne i granične uslove, kao i inverzne probleme uz neka dodatna merenja. Podržava domene složene geometrije i omogućava da korisnički kod bude kompaktan, veoma sličan matematičkoj formulaciji.

U odnosu na SCIANN, pristup DeepXDE podrazumeva nešto viši stepen apstrakcije, pa time i lakoće upotrebe. Na primer, komplikacije oko postavljanja kolokacionih tačaka u kojima važe granični i početni uslovi, kao i veličina *batch*-a, ne potpadaju pod brigu korisnika, već se sama biblioteka stara o tome da specificirani broj tačaka podleže graničnim uslovima. Taj nivo apstrakcije u nekim specifičnim slučajevima može predstavljati prepreku, ali u velikoj većini slučajeva doprinosi jasnijem definisanju problema.

Na sledećem listingu dati su značajni delovi implementacije:

Listing 3.1: Rešenje problema prigušenih oscilacija u 1D korišćenjem DeepXDE biblioteke

```
1  import deepxde as dde
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  m = 1
6  mu = 0.1
7  k = 2
8
9  # Pocetni uslovi
10 x0, v0 = 0, 2
11
12 delta = mu / (2*m)
13 w0 = np.sqrt(k/m)
14
15 # Da li je tacka blizu t=0 (provera pocetnog uslova)
16 def boundary_l(t, on_initial):
17     return on_initial and np.isclose(t[0], 0)
18
```

(continues on next page)

(nastavak sa prethodne stranice)

```
# Jednacina ODE
def Ode(t,x):
        dxdt = dde.grad.jacobian(x,t)
        dxdtt = dde.grad.hessian(x,t)
        return m * dxdtt + mu * dxdt + k*x

# x(0)=x0
def bc_func1(inputs, outputs, X):
    return outputs - x0

# x'(0)=v0
def bc_func2(inputs, outputs, X):
    return dde.grad.jacobian(outputs, inputs, i=0,j=None) - v0

# Resava se na domenu t=(0,10)
interval = dde.geometry.TimeDomain(0, 10)

# Pocetni uslovi
ic1 = dde.icbc.OperatorBC(interval, bc_func1, boundary_l)
ic2 = dde.icbc.OperatorBC(interval, bc_func2, boundary_l)

# Definisanje problema, granicnih uslova, broja kolokacionih tacaka
data = dde.data.TimePDE(interval, Ode, [ic1, ic2], 100, 20,␣
↪solution=func, num_test=100)

layers = [1] + [30] * 2 + [1]
activation = "tanh"
init = "Glorot uniform"
net = dde.nn.FNN(layers, activation, init)

model = dde.Model(data, net)

model.compile("adam", lr=.001, loss_weights=[0.01, 1, 1], metrics=["12␣
↪relative error"])
losshistory, train_state = model.train(epochs=30000)

T = np.linspace(0, 10, 100).reshape(100,1)
x_pred = model.predict(T)
```

Na početku se definišu importi i konstante problema, kao i dodatne konstante `delta` i `w0` koje smo koristili prilikom izvođenja analitičkog rešenja. Funkciju

```
def boundary_l(t, on_initial):
    return on_initial and np.isclose(t[0], 0)
```

ćemo iskoristiti za testiranje da li je data kolokaciona tačka blizu tačke `t=0`, tj. da li za

datu tačku važi početni uslov. Upotrebićemo je pri formiranju početnih uslova za poziciju i brzinu. Dalje, kao što naziv sugeriše, naredna funkcija

```
def Ode(t,x):
        dxdt = dde.grad.jacobian(x,t)
        dxdtt = dde.grad.hessian(x,t)
        return m * dxdtt + mu * dxdt + k*x
```

predstavlja postavku problema u svom izvornom obliku jednačine (3.1). Ovde je očigledna jedna od glavnih prednosti NMPFZ, tj. da neke dodatne transformacije u integracionim tačkama nisu potrebne, već samo postavka diferencijalne jednačine u vidu funkcije gubitka i graničnih uslova u istom obliku. Uslužne metode `dde.grad.jacobian` i `dde.grad.hessian` vraćaju prve, odnosno druge izvode po ulaznim varijablama primenjujući tzv. automatsku diferencijaciju. Podrazumevano se u pozadini koristi *Tensorflow* za tenzorske operacije niskog nivoa.

Postavka dva početna uslova u formi funkcije gubitka data je u sledeće dve metode, za koordinatu i brzinu respektivno:

```
def bc_func1(inputs, outputs, X):
    return outputs - x0

def bc_func2(inputs, outputs, X):
    return dde.grad.jacobian(outputs, inputs, i=0,j=None) - v0
```

Nakon postavke jednodimenzionog vremenskog domena u kome se problem rešava:

```
interval = dde.geometry.TimeDomain(0, 10)
```

možemo da formiramo i objekte graničnih uslova kombinujući funkcije gubitka sa funkcijom lokacije `boundary_l()`:

```
ic1 = dde.icbc.OperatorBC(interval, bc_func1, boundary_l)
ic2 = dde.icbc.OperatorBC(interval, bc_func2, boundary_l)
```

Sada imamo sve elemente da formiramo objekat problema koji rešavamo. Ovde ćemo to učiniti metodom `dde.data.TimePDE` za vremenski zavisne probleme:

```
data = dde.data.TimePDE(interval, Ode, [ic1, ic2], 100, 20,␣
↪solution=func, num_test=100)
```

Specificiramo redom računski domen, osnovnu jednačinu, listu graničnih uslova, broj kolokacionih tačaka za osnovni domen (100), broj kolokacionih tačaka za granične uslove (20), egzaktno rešenje (ako postoji) i broj testnih tačaka (za poređenje sa egzaktnim

rešenjem). U ovom primeru ćemo ignorisati egzaktno rešenje. Odmah se vidi razlika u postavci u odnosu na SCIANN pristup u načinu navođenja kolokacionih tačaka. Naime, kod DeepXDE kolokacione tačke se ne generišu manuelno, već se prepušta biblioteci da to uradi za nas, što ukazuje na jedan viši nivo apstrakcije.

Naredne linije koda konstruišu neuronsku mrežu koja će se koristiti kao aproksimacija problema, sa svim svojim hiper-parametrima:

```
layers = [1] + [30] * 2 + [1]
activation = "tanh"
init = "Glorot uniform"
net = dde.nn.FNN(layers, activation, init)
```

Naša mreža ima jedan ulaz, jedan izlaz i dva skrivena sloja od po 30 neurona, sa aktivacijom skrivenih slojeva u vidu `tanh` funkcije i odgovarajućom inicijalizacijom. Na kraju, možemo da krenemo u obučavanje, kada spojivši problem i generisanu neuronsku mrežu formiramo model:

```
model = dde.Model(data, net)
model.compile("adam", lr=.001, loss_weights=[0.01, 1, 1], metrics=["12␣
↪relative error"])
losshistory, train_state = model.train(epochs=30000)
```

Ovo su standardne metode koje se široko koriste u oblasti dubokog učenja, pa je dovoljno samo pomenuti da se navodi algoritam optimizacije, brzina učenja i način proračuna greške koja upravlja ovim procesom. Specifičnost za NMPFZ je što ovde listom `loss_weights` možemo i da „ponderišemo” težine osnovne diferencijalne jednačine, prvog i drugog graničnog uslova, respektivno. U narednoj sekciji *Rešenja direktnog problema* (stranica 44) ćemo razmotriti rešenja za sva tri slučaja prigušenog oscilovanja.

3.3 Rešenja direktnog problema

Sada ćemo se pozabaviti rezultatima procesa učenja koji smo uspostavili u prethodnoj sekciji *Implementacija* (stranica 40).

3.3.1 Podprigušeni slučaj

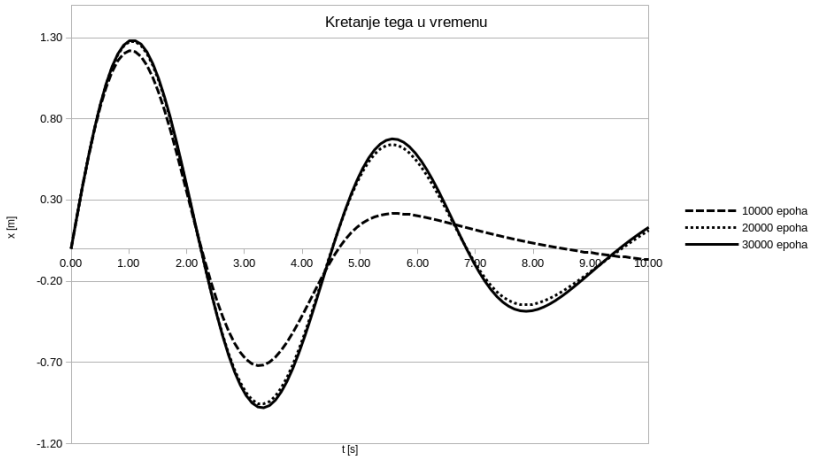
Za slučaj da je:

$$\begin{aligned} m &= 1 \\ \mu &= 0,1 \\ k &= 2 \end{aligned}$$

imamo da su:

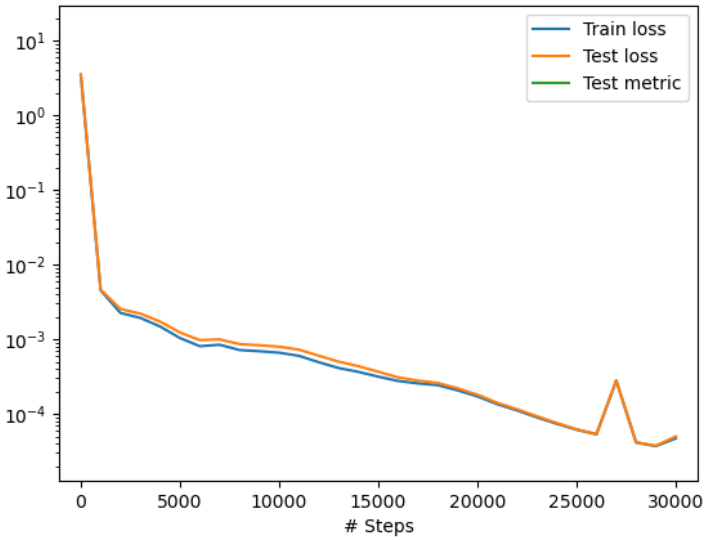
$$\begin{aligned} \delta &= \frac{\mu}{2m} = 0,05 \\ \omega_0 &= \sqrt{\frac{k}{m}} = \sqrt{2} \end{aligned}$$

Kako je $\delta^2 - \omega_0^2 < 0$, imaćemo dva konjugovano-kompleksna rešenja, tj. podprigušeni slučaj opisan u sekciji *Podprigušeni slučaj* (stranica 38). Rešenja dobijena skriptom datom u *Implementacija* (stranica 40), gde su početni uslovi postavljeni tako da je $(x_0 = 0, v_0 = 2)$ prikazana su na Sl. 3.5.



Sl. 3.5: NMPFZ rešenje promene položaja tega u toku vremena za podprigušeni slučaj oscilovanja.

Na grafiku se može videti kako se rezultati razlikuju u odnosu na to koliko epoha je mreža trenirana. Naime, rezultati za 10000 epoha su značajno lošiji nego oni za 20000 i 30000 epoha. Dakle, kao i kod gotovo svih problema dubokog učenja i kod NMPFZ taj proces treba pratiti (Sl. 3.6) i trening prekinuti tek kada je dosegnut odgovarajući minimum i učenje dalje ne napreduje značajno.



Sl. 3.6: Funkcija gubitka u toku procesa učenja.

3.3.2 Preprigušeni slučaj

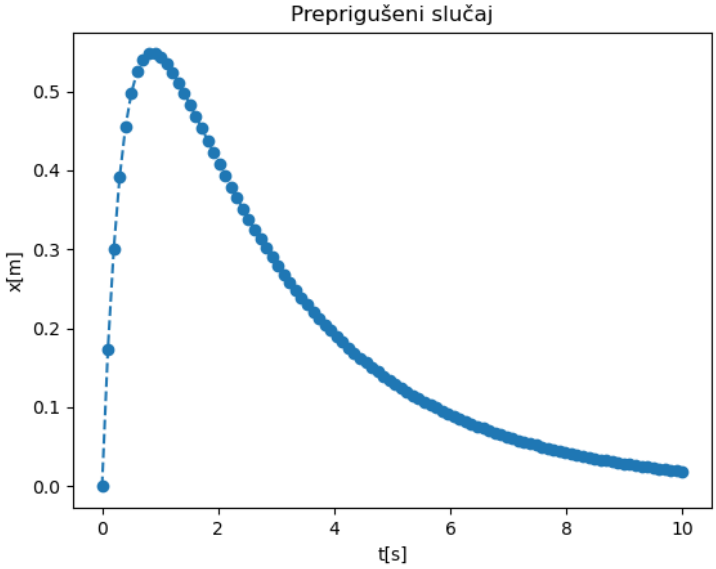
Ukoliko je, na primer:

$$m = 1$$
$$\mu = 3$$
$$k = 1$$

imamo da su:

$$\delta = \frac{\mu}{2m} = 1,5$$
$$\omega_0 = \sqrt{\frac{k}{m}} = 1$$

Kako je $\delta^2 - \omega_0^2 > 0$, imaćemo dva različita realna rešenja, tj. preprigušeni slučaj opisan u sekciji *Preprigušeni slučaj* (stranica 36). Rešenja dobijena skriptom datom u *Implementacija* (stranica 40) za početne uslove $(x_0 = 0, v_0 = 2)$ prikazana su na Sl. 3.7.



Sl. 3.7: NMPFZ rešenje promene položaja tega u toku vremena za preprigušeni slučaj oscilovanja.

3.3.3 Kritično-prigušeni slučaj

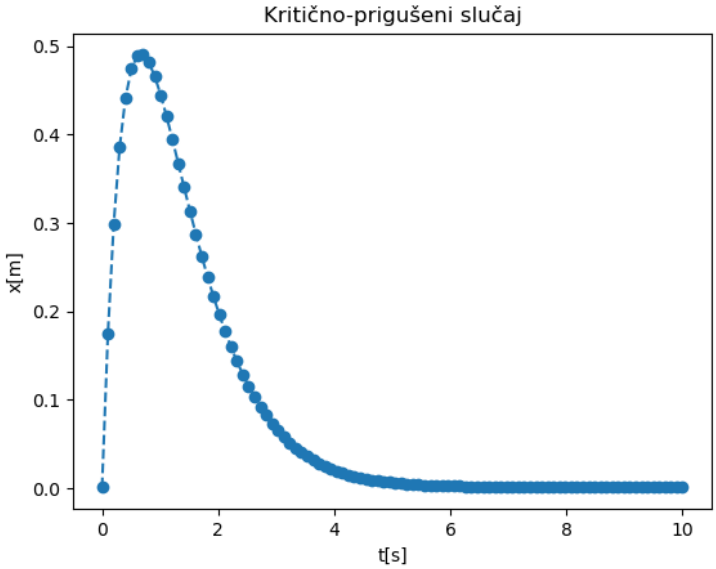
Preostao je još kritično-prigušeni slučaj, koji će se dobiti ukoliko postavimo sledeće parametre problema:

$$m = 1$$
$$\mu = 3$$
$$k = 2,25$$

Imamo da su:

$$\delta = \frac{\mu}{2m} = 1,5$$
$$\omega_0 = \sqrt{\frac{k}{m}} = 1,5$$

Kako je $\delta^2 - \omega_0^2 = 0$, imaćemo dva jednaka realna rešenja, tj. kritično-prigušeni slučaj opisan u sekciji *Kritično-prigušeni slučaj* (stranica 37). Rešenja dobijena skriptom datom u *Implementacija* (stranica 40) za početne uslove $(x_0 = 0, v_0 = 2)$ prikazana su na Sl. 3.8.



Sl. 3.8: NMPFZ rešenje promene položaja tega u toku vremena za kritično-prigušeni slučaj oscilovanja.

3.4 Inverzni problem

Kao što smo već u više navrata naglasili, prava snaga NMPFZ dolazi do izražaja kod problema u kojima je potrebno identifikovati nepoznate parametre. Razlog tome je što NMPFZ metoda inverzne probleme tretira na isti način kao direktne. Važno je samo da se problem postavi ne dozvoljavajući višeznačnost, što ćemo demonstrirati na sledećem primeru. Zamislimo eksperiment postavljen kao na Sl. 3.1 u kome nam je poznat samo koeficijent trenja $\mu = 0,6$, a ne znamo ni masu kuglice m ni koeficijent elastičnosti opruge k. Zadatak nam je da identifikujemo ova dva parametra tako što ćemo pustiti kuglicu da osciluje i na osnovu njenog kretanja dati NMPFZ da odredi elemente koji nedostaju.

Recimo da smo vizuelno utvrdili da je u pitanju **podprigušeni slučaj**. Prva strategija

merenja koja bi mogla da bude realno izvodljiva je da kuglicu otklonimo za $x_0 = 2$ i samo pustimo ($v_0 = 0$), a onda štopericom odredimo vremenske trenutke u kojima kuglica prolazi kroz ravnotežni položaj i da te trenutke zabeležimo:

t	1,18	3,27	5,37	7,46	9,55
x	0	0	0	0	0

Dakle, pored graničnih i početnih uslova ($x_0 = 2, v_0 = 0$) uvešćemo i granične uslove tipa `PointSet` koji definišu vrednost modelovane funkcije u pojedinim tačkama. Programski kod koji implementira ovaj inverzni problem dat je na sledećem listingu.

Listing 3.2: Inverzni problem prigušenih oscilacija u 1D. Nepoznati parametri su `m` i `k`.

```
import deepxde as dde
from deepxde.backend import tf
import numpy as np
import matplotlib.pyplot as plt

m = dde.Variable(0.5)
mu = 0.6
k = dde.Variable(2.)

x0, v0 = 2, 0

delta = mu / (2*m)
w0 = tf.sqrt(k/m)

# Da li je tacka blizu t=0 (provera pocetnog uslova)
def boundary_l(t, on_initial):
    return on_initial and np.isclose(t[0], 0)

# Jednacina ODE
def Ode(t,x):
        dxdt = dde.grad.jacobian(x,t)
        dxdtt = dde.grad.hessian(x,t)
        return m * dxdtt + mu * dxdt + k*x

# x(0)=x0
def bc_func1(inputs, outputs, X):
    return outputs - x0

# x'(0)=v0
def bc_func2(inputs, outputs, X):
    return dde.grad.jacobian(outputs, inputs, i=0,j=None) - v0
```

(continues on next page)

(nastavak sa prethodne stranice)

```
32
33 # Resava se na domenu t=(0,10)
34 interval = dde.geometry.TimeDomain(0, 10)
35
36 # Pocetni uslovi
37 ic1 = dde.icbc.OperatorBC(interval, bc_func1, boundary_l)
38 ic2 = dde.icbc.OperatorBC(interval, bc_func2, boundary_l)
39
40 bc_x = np.array([1.18, 3.27, 5.37, 7.46, 9.55]).reshape(6,1)
41 bc_y = np.array([0, 0, 0, 0, 0 ]).reshape(6,1)
42 ic3 = dde.icbc.PointSetBC(bc_x, bc_y, component=0)
43
44 # Definsanje problema, granicnih uslova, broja kolokacionih tacaka
45 data = dde.data.TimePDE(interval, Ode, [ic1, ic2, ic3], 200, 20,␣
   ↪solution=func, num_test=100)
46
47 layers = [1] + [30] * 3 + [1]
48 activation = "tanh"
49 init = "Glorot uniform"
50 net = dde.nn.FNN(layers, activation, init)
51
52 model = dde.Model(data, net)
53
54 # Callback funkcija koja stampa varijablu na svakih 1000 epoha
55 variable1 = dde.callbacks.VariableValue(k, period=1000)
56 variable2 = dde.callbacks.VariableValue(m, period=1000)
57
58 model.compile("adam", lr=.001, loss_weights=[0.01, 1, 1, 1], metrics=[
   ↪"l2 relative error"], external_trainable_variables=[k,m])
59 losshistory, train_state = model.train(epochs=50000,␣
   ↪callbacks=[variable1, variable2])
```

Objasnićemo samo delove koji se razlikuju u odnosu na direktni problem rešen u sekciji *Implementacija* (stranica 40). Za početak, tu su nepoznati parametri koje deklarišemo na sledeći način:

```
m = dde.Variable(0.5)
k = dde.Variable(2.)
```

U zagradi se daju početne vrednosti parametra. Sledeće linije definišu pomenuti dodatni `PointSet` granični uslov (uslove) koji važe u pojedinim tačkama *unutar domena*:

```
bc_x = np.array([1.18, 3.27, 5.37, 7.46, 9.55]).reshape(6,1)
bc_y = np.array([0, 0, 0, 0, 0 ]).reshape(6,1)
ic3 = dde.icbc.PointSetBC(bc_x, bc_y, component=0)
```

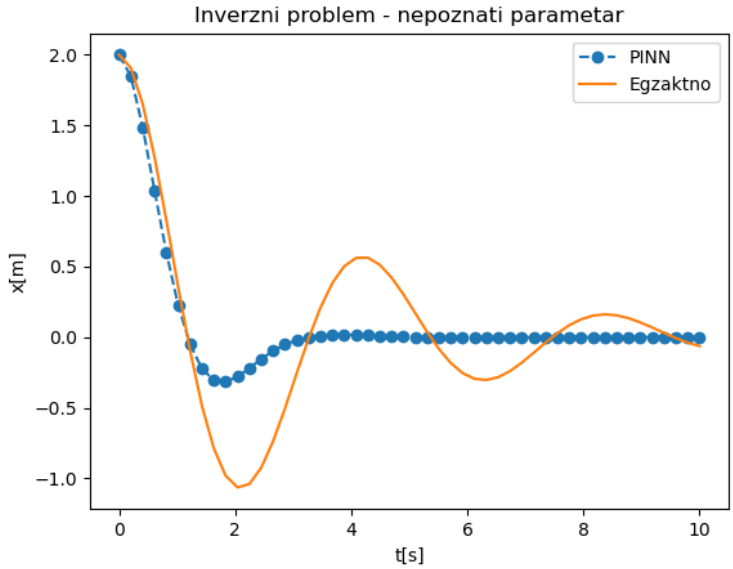
Kako bismo obezbedili praćenje vrednosti nepoznatih parametara tokom obuke, potrebno je da postavimo tzv. `callback` funkcije, koje će se pozivati na svakih 1000 epoha:

```
variable1 = dde.callbacks.VariableValue(k, period=1000)
variable2 = dde.callbacks.VariableValue(m, period=1000)
```

Prilikom postavljanja modela postavljamo eksterne varijable za treniranje `k` i `m`, dok se pri pozivu treninga navode `callback` funkcije:

```
model.compile("adam", lr=.001, loss_weights=[0.01, 1, 1, 1], metrics=[
↪"l2 relative error"], external_trainable_variables=[k,m])
losshistory, train_state = model.train(epochs=50000,␣
↪callbacks=[variable1, variable2])
```

Nakon završenog obučavanja, dobija se, očigledno pogrešno, rešenje koje je prikazano na Sl. 3.9. U odnosu na analitičko rešenje koje je postavljeno koristeći vrednosti parametara `m=1` i `k=2,25`, dobijene vrednosti `m=0,287` i `k=1,22` se puno razlikuju.



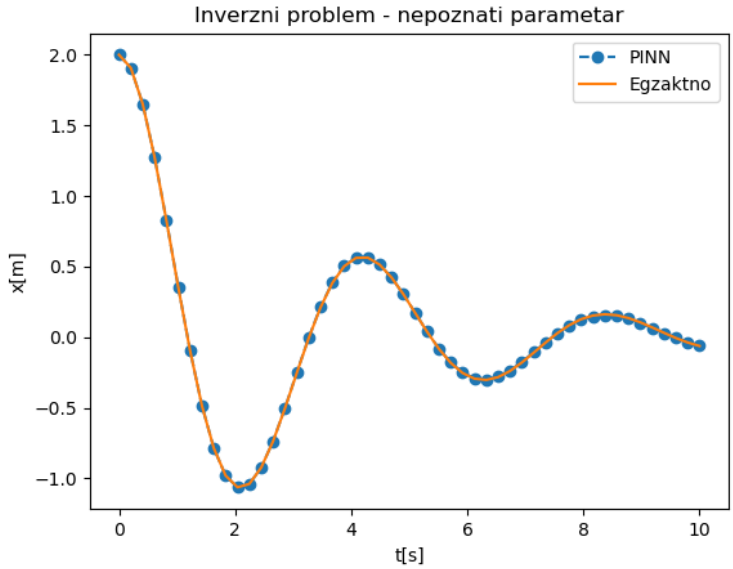
Sl. 3.9: NMPFZ rešenje inverznog problema sa nepoznatim parametrima. Pronađene vrednosti su `m=0,287` i `k=1,22`.

Zašto smo dobili ovako loše rešenje? Odgovor se krije u loše postavljenim graničnim uslovima koji dovode do nejednoznačnosti inverznog problema. Naime, postoji više sce-

narija, tj. parova `(m,k)` koji zadovoljavaju granične uslove postavljene samo u tačkama prolaska tega kroz ravnotežni položaj. Očigledno je da moramo da dodamo još neku tačku van ravnotežnog položaja, kako bismo obezbedili jednoznačno rešenje. Zamislimo da smo izmerili i trenutak kada je teg bio na najvećoj negativnoj udaljenosti u odnosu na ravnotežni položaj i kolika je ta udaljenost bila. Dodajmo sada i tu tačku u postavljeni `PointSet`, koji sada izgleda ovako:

t	1,18	3,27	5,37	7,46	9,55	2,12
x	0	0	0	0	0	-1,67

Pogledom na Sl. 3.10 odmah se vidi da je poklapanje sa analitičkim rešenjem u ovako postavljenom problemu skoro pa idealno.



Sl. 3.10: NMPFZ rešenje inverznog problema sa nepoznatim parametrima. Pronađene vrednosti su `m=0,98` i `k=2,26`.

Ispravnost rešenja dodatno potvrđuju parametri `m=0,98` i `k=2,26`, čije su vrednosti veoma bliske onima koje su date u analitičkoj postavci `m=1` i `k=2,25`. Na ovaj način smo pokazali da pristup rešavanju inverznog problema, iako metodološki sličan, ima specifičnosti o kojima treba voditi računa. Kod direktnih problema rešenje je uvek jednoznačno,

dok kod inverznih moraju da se obezbede odgovarajući uslovi koji u dovoljnoj meri determinišu rešenje.

4 Hidrologija

4.1 Uvod

U ovom poglavlju bavićemo se hidrološkim problemima koje su autor i saradnici imali prilike da rešavaju u praksi. Poglavlje se sastoji iz dva rešena primera. Prvi primer se bavi strujanjem podzemnih voda, dok se drugi bavi problematikom predviđanja ponašanja poplavnog talasa u otvorenim tokovima. Oba problema rešavaćemo pomoću NMPFZ i usput naglašavati razlike u odnosu na tretman metodom konačnih elemenata, koja je u ovoj oblasti *de-facto* standard.

4.2 Strujanje podzemnih voda

Osnovna veličina od koje se polazi u teoriji strujanja tečnosti kroz porozno tlo je potencijal ϕ definisan kao

$$\phi = \frac{p}{\gamma} + h,$$

gde je p pritisak tečnosti, γ specifična težina, a h visina merena u odnosu na izabranu referentnu ravan. Brzina tečnosti $\mathbf{q}$, poznata i kao Darsijeva brzina, predstavlja zapreminu tečnosti koja prođe u jedinici vremena kroz jediničnu površinu porozne sredine. Ona se može izraziti pomoću potencijala ϕ relacijom koja se zove Darsijev zakon:

$$\mathbf{q} = -\mathbf{K}\nabla\phi,$$

gde je **K** matrica permeabilnosti koja za ortotropni materijal ima oblik

$$\mathbf{K} = \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & k_z \end{bmatrix},$$

gde su k_x, k_y i k_z koeficijenti permeabilnosti u odgovarajućim pravcima. Komponentni oblik jednačine je prema tome:

$$\begin{aligned} q_x &= -k_x \frac{\partial \phi}{\partial x}, \\ q_y &= -k_y \frac{\partial \phi}{\partial y}, \\ q_z &= -k_z \frac{\partial \phi}{\partial z}. \end{aligned}$$

Sada u sistem uvodimo jednačinu kontinuiteta. U slučaju stacionarnog strujanja nestišljivog fluida, kakva je voda, jednačina kontinuiteta ima oblik

$$\nabla^T \mathbf{q} = 0,$$

ili uz korišćenje Darsijevog zakona

$$\frac{\partial}{\partial x}\left(k_x \frac{\partial \phi}{\partial x}\right) + \frac{\partial}{\partial y}\left(k_y \frac{\partial \phi}{\partial y}\right) + \frac{\partial}{\partial z}\left(k_z \frac{\partial \phi}{\partial z}\right) = 0$$

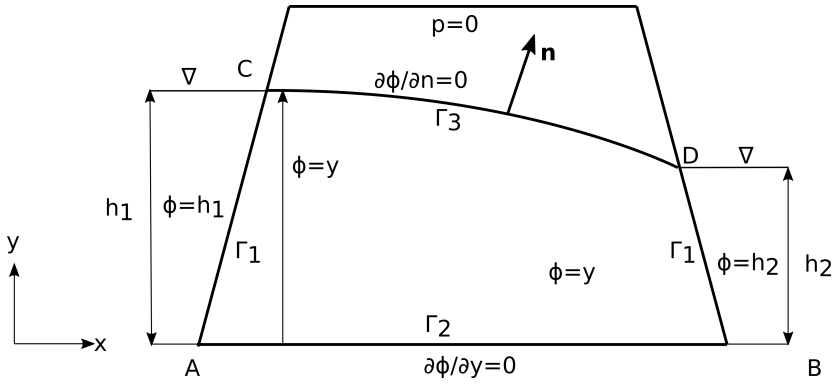
U slučaju kada se promena koeficijenata k_x, k_y, k_z sa koordinatama može zanemariti, što je najčešći slučaj, jednačina se svodi na

$$k_x \frac{\partial^2 \phi}{\partial x^2} + k_y \frac{\partial^2 \phi}{\partial y^2} + k_z \frac{\partial^2 \phi}{\partial z^2} = 0$$

Konačno, ako postoji izvor i/ili ponor, za stacionarne uslove, **hidrodinamička jednačina** ima sledeći oblik:

$$k_x \frac{\partial^2 \phi}{\partial x^2} + k_y \frac{\partial^2 \phi}{\partial y^2} + k_z \frac{\partial^2 \phi}{\partial z^2} + \bar{Q} = 0 \tag{4.1}$$

gde je $\bar{Q}$ zapreminski fluks (izvor ili ponor, kao količina tečnosti po jedinici zapremine porozne sredine u jedinici vremena). Granični uslovi koji se sreću u rešavanju problema strujanja kroz poroznu sredinu opisanog gornjim jednačinama prikazani su na Sl. 4.1.



Sl. 4.1: Različiti granični uslovi kod problema filtracije u dve dimenzije.

Oni mogu biti:

1. **zadati potencijal**

$$\phi = \bar{\phi}, \qquad | \, \Gamma_1$$

2. **zadati površinski protok (fluks)**

$$q_n = \bar{q} \qquad | \, \Gamma_2$$

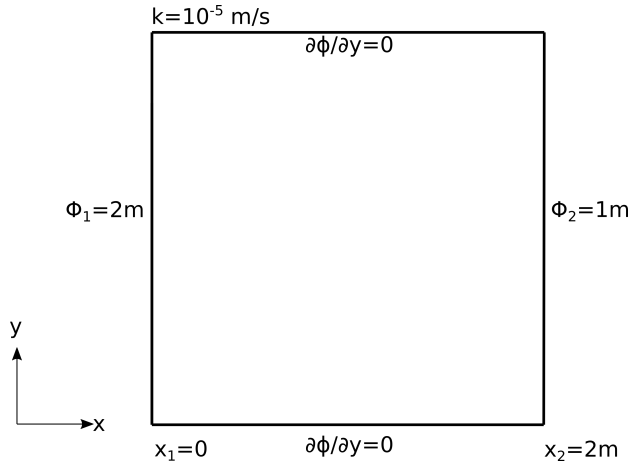
3. **slobodna površina**

$$p = 0, \, \phi = y, \, \frac{\partial \phi}{\partial n} = 0 \qquad | \, \Gamma_3 \tag{4.2}$$

Primetimo da je na slobodnoj površini $\phi = y$. Pošto se oblik slobodne površine ne zna, to je njeno određivanje poseban zadatak u ovoj oblasti. I ovaj problem ćemo pokušati da pokrijemo metodom NMPFZ.

4.2.1 Stacionarno strujanje kroz poroznu sredinu

Dvodimenzionalno stacionarno tečenje kroz porozni medijum je regulisano konstantnom razlikom potencijala na dve površine. Protok se javlja između dva nepropusna sloja u pravougaonoj geometriji dimenzija *2m x 2m*, kao na Sl. 4.2.



Sl. 4.2: Postavka problema stacionarnog strujanja kroz poroznu sredinu bez slobodne površine

Implementacija problema je jednostavna i njeni najvažniji delovi se nalaze na sledećem listingu.

Listing 4.1: Rešenje problema strujanja bez slobodne površine u 2D korišćenjem SCIANN biblioteke

```
# Osnovni grid
x_data, y_data = np.meshgrid(
    np.linspace(0, 2, 201),
    np.linspace(0, 2, 201)
)

# Modeluje se phi(x,y)
x = sn.Variable('x')
y = sn.Variable('y')
phi = sn.Functional('phi', [x,y], 4*[30], 'sigmoid')

# %%
k = 1.e-5
TOL = 0.015

# Osnovna jednacina
fun1 = k * (diff(phi, x, order=2) + diff(phi, y, order=2))

# Dirihleovi granicni uslovi
C1 = (1-sign(x - (0+TOL))) * (phi-2)
```

(continues on next page)

(nastavak sa prethodne stranice)

```
C2 = (1+sign(x - (2-TOL))) * (phi-1)

# Njumanovi granicni uslovi
N1 = (1-sign(y - (0+TOL))) * diff(phi,y)
N2 = (1+sign(y - (2-TOL))) * diff(phi,y)

# FZNN model
m2 = sn.SciModel([x,y], [fun1, C1, C2, N1, N2],  optimizer='Adam')

# Trening
pinn_model = m2.train([x_data, y_data], 5*['zero'], learning_rate=0.
↪001, batch_size=1024, epochs=100, stop_loss_value=1E-15)
```

Sa svim ovim postavkama smo se manje-više već sretali, osim što do sada nismo imali 2D stacionarni problem. Postavljamo ravnomernu mrežu kolokacionih tačaka u dimenzijama domena (*2m x 2m*), zatim definišem funkcional $\Phi(x, y)$ i diferencijalnu jednačinu problema. Primetimo da rešenje uopšte ne bi trebalo da zavisi od koeficijenta k. Sledeći korak je postavka Dirihleovih graničnih uslova na levom ($\Phi_1 = 2m$) i na desnom ($\Phi_1 = 1m$) kraju domena, tj. na vertikalama $x_1 = 0m$ i $x = 2m$, respektivno:

```
C1 = (1-sign(x - (0+TOL))) * (phi-2)
C2 = (1+sign(x - (2-TOL))) * (phi-1)
```

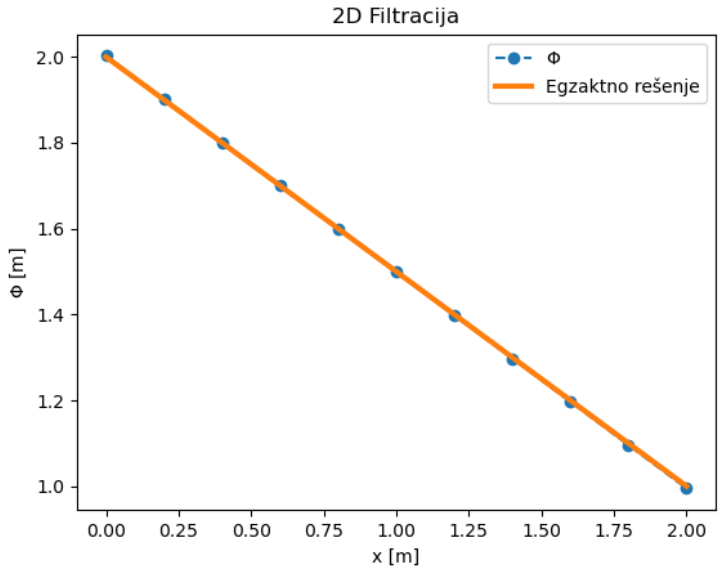
Nedostaju samo još Nojmanovi granični uslovi koji jamče da su donja ($y = 0$) i gornja ($y = 2m$) površina nepropusne, tj. da je izvod potencijala po y jednak nuli:

```
N1 = (1-sign(y - (0+TOL))) * diff(phi,y)
N2 = (1+sign(y - (2-TOL))) * diff(phi,y)
```

Kada se postavi problem, rešenje se nazire već za nekoliko desetina epoha treniranja. Analitičko rešenje za potencijal je, prema Bear [Bea12]:

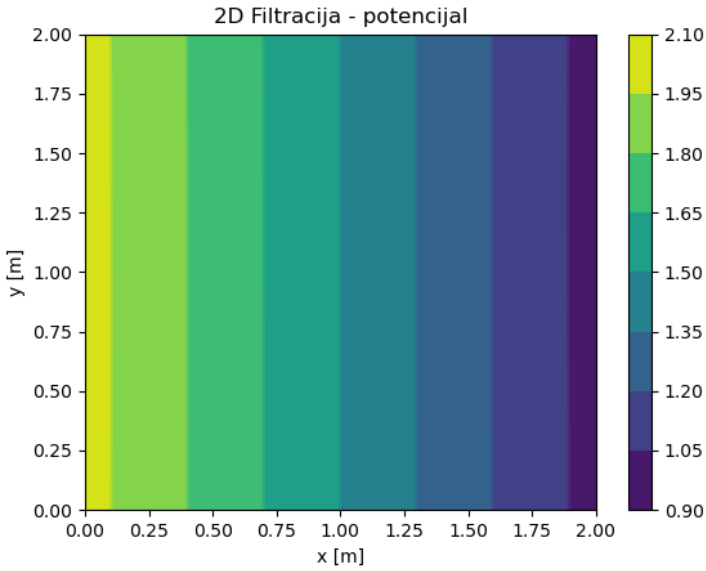
$$\phi = \Phi_1 - \frac{\Phi_1 - \Phi_2}{L}(x - x_1)$$

gde je $L = x_2 - x_1$. Dakle, polje potencijala je konstantno u odnosu na Y osu, dok je gradijent potencijala konstantan u pravcu X ose.



Sl. 4.3: NMPFZ rešenje potencijala duž X ose za 2D slučaj strujanja bez slobodne površine.

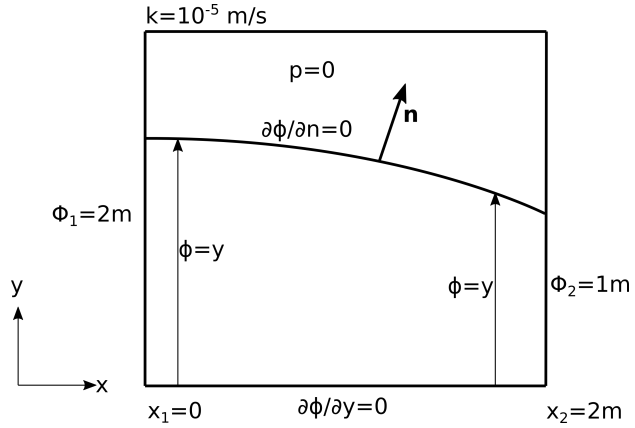
Poređenje analitičkog i NMPFZ rešenja je prikazano na Sl. 4.3, a polje potencijala je prikazano na Sl. 4.4. Uniformnost potencijalnog polja u Y smeru, dodatno potvrđuje tačnost 2D NMPFZ rešenja za ovaj stacionarni problem.



Sl. 4.4: NMPFZ rešenje polja potencijala za 2D slučaj strujanja bez slobodne površine.

4.2.2 Stacionarno strujanje kroz poroznu sredinu sa slobodnom površinom

Stacionarno tečenje kroz porozni medijum, sa slobodnom površinom je regulisano konstantnom razlikom potencijala na dve suprotne površine, kao što je prikazano na Sl. 4.5. Donja površina je nepropusna. Geometrijski i materijalni podaci, kao i granični uslovi, takođe su dati na Sl. 4.5.



Sl. 4.5: Postavka problema stacionarnog strujanja kroz poroznu sredinu sa slobodnom površinom

Vrednosti potencijala u kolokacionim tačkama na površini $x_1 = 0$ su $\Phi_1 = 2m$ dok su u tačkama na liniji $x_2 = 2m$ vrednosti $\Phi_2 = 1m$. Donja površina je nepropusna, pa na njoj zadajemo da je gradijent potencijala nula. Dakle, i Dirihleovi i Nojmanovi granični uslovi su identični kao i u prethodnom primeru koji nije uključivao postojanje slobodne površine. Međutim, njeno postojanje je fizički nužno i definisano uslovima (4.2).

Kako bismo implementirali ovaj granični uslov, moramo da izračunamo pravac normale:

```
k1 = diff(phi,x)
alpha = atan(k1)+np.pi/2
nx = cos(alpha)
ny = sin(alpha)
```

koji ćemo dobiti time što dodamo ugao $\frac{\pi}{2}$ pravcu tangente na `phi`, koju izračunavamo zahvaljujući trivijalnoj dostupnosti prvog izvoda u NMPFZ metodologiji. Nakon toga lako izračunavamo komponente normale `nx` i `ny`. Granični uslov slobodne površine postavljamo na isti način kao i ranije kada smo koristili biblioteku SCIANN, tako što se u vidu konjunkcije navodi gde uslov važi i šta u tom delu domena važi. Međutim, ovog puta nemamo strogo definisane koordinate, jer položaj slobodne površine ne znamo. Ono što znamo je da je na čitavoj slobodnoj površini $\phi = y$, pa ovo navodimo kao oblast važenja:

```
FS1 = (abs(y-phi)<0.009) * k * (diff(phi,x)*nx + diff(phi,y)*ny)
```

dok uslov nepostojanja protoka kroz slobodnu površinu $\frac{\partial \phi}{\partial n} = \frac{\partial \phi}{\partial x} n_x + \frac{\partial \phi}{\partial y} n_y = 0$ navodimo kao glavnu komponentu.

Potrebno je obezbediti i dovoljan broj kolokacionih tačaka da bi se ispravno ispratio oblik slobodne površine. To ćemo obezbediti tako što u delu domena u kome očekujemo pojavu slobodne površine koncentraciju kolokacionih tačaka povećamo (u našoj implementaciji četiri puta). Kako je u pitanju čisto tehničko rešenje, ovde se time nećemo baviti, već čitaoca upućujemo na kompletan primer.

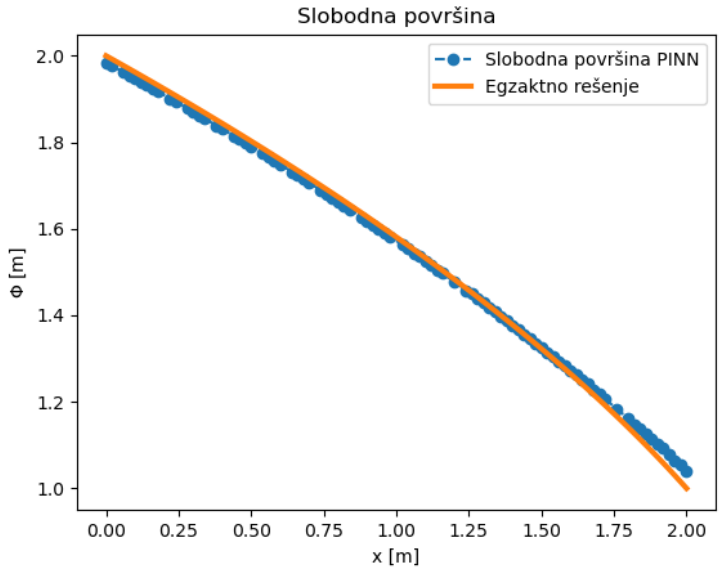
Analitičko rešenje za potencijal za ovaj jednostavan problem se po Bear [Bea12], može se napisati u obliku

$$\phi = \sqrt{\Phi_1^2 - 2B(x - x_1)},$$

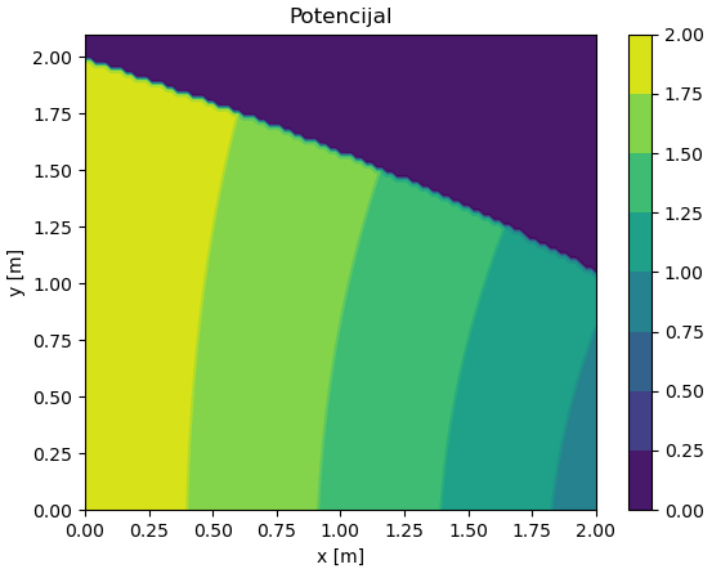
gde je

$$B = \frac{\Phi_1^2 - \Phi_2^2}{2L}$$

i $L = x_2 - x_1$. Poređenje NMPFZ rešenja sa analitičkim rešenjem može se videti na Sl. 4.6. Polje potencijala je prikazano na Sl. 4.7.



Sl. 4.6: NMPFZ rešenje potencijala duž X ose za 2D slučaj strujanja sa slobodnom površinom.



Sl. 4.7: NMPFZ rešenje polja potencijala za 2D slučaj strujanja sa slobodnom površinom.

Može se primetiti relativno dobro slaganje NMPFZ rešenja sa analitičkim rešenjem, kao i očigledna razlika rasporeda polja potencijala u odnosu na slučaj bez slobodne površine prikazan na Sl. 4.4. Ako pak uporedimo pristup rešavanju problema slobodne površine metodom NMPFZ sa klasičnom metodom konačnih elemenata kod Kojić [Kojic98], možemo primetiti da je NMPFZ pristup jednostavniji. Razlog tome je što se kod NMPFZ ne zahteva nikakav poseban numerički tretman i upotreba numeričkih pretpostavki, već se fizika problema direktnim putem prevodi u NMPFZ granični uslov.

4.3 Propagacija talasa u otvorenom kanalu

Upravljanje vodnim resursima zahteva alate za dugoročno i kratkoročno predviđanje različitih hidroloških fenomena. Predviđanje je značajno u oblastima upravljanja rizikom od poplava, upravljanja hidro-elektranama, unutrašnje plovidbe, vodosnabdevanja, itd. Brojni zadaci u vezi sa predviđanjem hidroloških podataka uspešno su obrađeni korišćenjem pristupa predviđanja zasnovanog na klasičnom numeričkom modelovanju. Iako ovaj pristup daje zadovoljavajuće rezultate, još uvek ima mnogo problema koje treba rešiti. U mnogim slučajevima, vreme izračunavanja je parametar koji može ograničiti primenu fizički zasnovanih hidroloških i hidrauličkih modela u realnoj primeni. Pored toga, fizički

zasnovani modeli su nedovoljno fleksibilni u slučajevima inverznih problema, identifikacije parametara, asimilacije merenih podataka itd.

Modeli predviđanja zasnovani na neuronskim mrežama primenjeni su u modelovanju padavina i oticaja Chadalawada *et al.* [CHB20], sistemima ranog upozorenja na poplave Duncan *et al.* [DCK+13], kao i modelovanju urbanih vodovodnih mreža Garzón *et al.* [GarzonKLT22]. Ovi pristupi zahtevaju veliku količinu podataka za obuku, što može stvoriti probleme ukoliko nema dovoljno podataka. Pored toga, može se primetiti da ova vrsta modela nije u stanju da isporuči dobre rezultate kada ulazni podaci izlaze izvan opsega podataka koji se koriste za obuku. U ovim slučajevima, modeli zasnovani na neuronskim mrežama mogu proizvesti fizički nemoguće rezultate. U poslednjih nekoliko godina, ubrzano se uvodi NMPFZ pristup i u ovoj oblasti. U narednom primeru ćemo sagledati potencijal upotrebe NMPFZ u jednodimenzionom problemu propagacije poplavnog talasa u otvorenim kanalima spajanjem fizičkog zakona sa početnim i graničnim uslovima opisanim kinematičkom jednačinom propagacije talasa.

4.3.1 Usmeravanje poplava - fizički zakoni

Prostiranje poplavnog talasa u otvorenim kanalima je opisano pomoću dve jednačine, i to **jednačinom kontinuiteta** (4.3) i **zakonom održanja količine kretanja** (4.4). Ova jednačina sadrži uticaje trenja, gravitacije, sile pritiska, kao i lokalnog i konvektivnog ubrzanja. U ovom primeru, prostiranje talasa u pravougaonom kanalu je predstavljeno kinematičkim talasom koji pojednostavljuje dinamičku jednačinu, uzimajući samo uticaj trenja i gravitacije:

$$\frac{\partial h(x,t)}{\partial t} + c\frac{\partial h(x,t)}{\partial x} = 0 \tag{4.3}$$

$$Q(x,t) = \frac{1}{n} \cdot B \cdot h(x,t)^{\frac{5}{3}} \cdot \sqrt{I_d}, \tag{4.4}$$

gde $h\,[m]$ predstavlja dubinu vode, $t\,[s]$ je vreme, $x\,[m]$ prostornu koordinatu, $c\,[m/s]$ brzinu propagacije poremećaja, $Q[m^3/s]$ protok, $n[m^{-\frac{1}{3}}s]$ reprezentuje Maningovu hrapavost, $B\,[m]$ širinu poprečnog preseka i I_d nagib po uzdužnoj osi.

Cilj modelovanja prostiranja talasa je procena promene dubine vode duž kanala $h(x,t)$ koja je izazvana poplavnim talasom predstavljenim hidrogramom toka $Q_{in}(t) = Q(0,t)$ na uzvodnom kraju kanala.

4.3.2 Konstrukcija funkcije gubitka

Ako se osvrnemo na opšti izraz za funkciju gubitka (1.2), vidimo tri komponente, i to gubitak koji potiče od rezidualne mreže, gubitak koji potiče od početnih uslova i gubitak koji potiče od graničnih uslova. Krenimo redom. Komponenta funkcije gubitka koja potiče od diferencijalne jednačine (4.3) definisana je desnom stranom iste jednačine. Komponenta početnih uslova se definiše kao:

$$h(x, t = 0) = h_0$$

Pored toga, potreban nam je i granični uslov koji definiše vrednost visine talasa na početku modelovanog domena, tj. na $x = 0$ odakle talas dolazi. Taj granični uslov izvodimo iz dinamičke jednačine (4.4):

$$h(0, t) = \left(\frac{Q_{in}(t) \cdot n}{B \cdot \sqrt{I_d}} \right)^{\frac{3}{5}}.$$

Ukoliko kao meru greške usvojimo srednju kvadratnu grešku (*Mean Squared Error - MSE*), kompozitna funkcija gubitka izgledaće ovako:

$$MSE = MSE_r + MSE_0 + MSE_b,$$

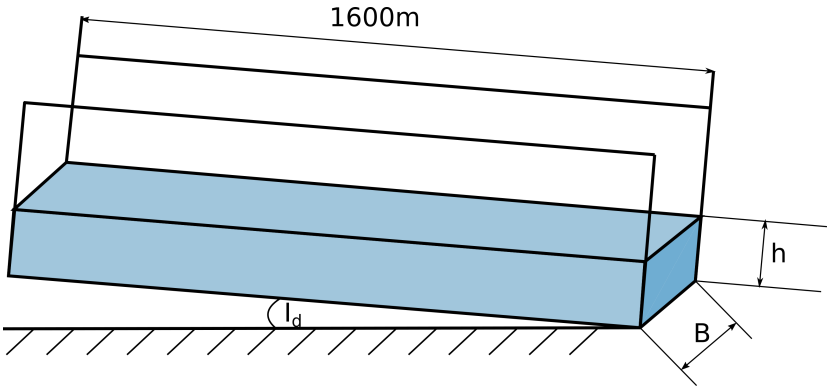
gde su:

$$MSE_r = \frac{1}{N_{x_f,t_f}} \sum |r(x_f, t_f)|^2,$$
$$MSE_0 = \frac{1}{N_{x_0,t_0}} \sum \left| \tilde{h}(x_0, 0) - h(x_0, 0) \right|^2,$$
$$MSE_b = \frac{1}{N_{x_b,t_b}} \sum \left| \tilde{h}(0, t_b) - h(0, t_b) \right|^2.$$

Ovde su N_{x_f,t_f}, N_{x_0,t_0} i N_{x_b,t_b} ukupni brojevi kolokacionih tačaka u unutrašnjosti modelovanog domena, za početne i za granične uslove, respektivno.

4.3.3 Test primer i implementacija

Primer na kome ćemo testirati valjanost našeg NMPFZ pristupa za modelovanje širenja poplavnog talasa je propagacija talasa duž kanala dugog 1600 metara, oblika prizme i pravougaonog poprečnog preseka širokog 15 metara, kao na Sl. 4.8.



Sl. 4.8: Postavka problema propagacije poplavnog talasa u vremenu.

Maningova hrapavost ima vrednost od $n = 0,015 m^{-\frac{1}{3}} s$, nagib je postavljen na $I_d = 0,005$, a brzina propagacije na $c = 15\, m/s$. Cilj je izračunati promene dubine i protoka vode duž kanala, izazvane poplavnim talasom generisanim kao uzvodni granični uslov na $x = 0$:

$$Q_{in}(t) = Q(0,t) = 180 \cdot \left[1 + \left(-\frac{sgn(t-600)}{2} + \frac{1}{2}\right) \cdot \sin\left(\frac{\pi t}{600}\right)\right].$$

Pored ovog uslova, tu je i početni uslov Dirihleovog tipa, a to je da je visina vode u kanalu $h(x, t = 0) = 1,751m$. Interesantni delovi rešenja prikazani su na Listing 4.2.

Listing 4.2: Rešenje problema propagacije poplavnog talasa korišćenjem DeepXDE okvira

```
import deepxde as dde
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from deepxde.backend import tf

c = 15 # brzina propagacije talasa
n = 0.015 # hrapavost kanala
Id = 0.005 # nagib dna kanala
B = 15 # poprecni presek
length = 1600
total_time = 1000.0

# Hiperparametri
layers = [2] + [30] * 4 + [1]
activation = 'tanh'
```

(continues on next page)

(nastavak sa prethodne stranice)

```
initializer = 'Glorot uniform'
optimizer = 'rmsprop'
batch_size = 128
num_of_epochs = 20000
learning_rate = 0.001
loss = 'mse'

# Jednacina kontinuiteta
def pde(x, h):
    dh_t = dde.grad.jacobian(h, x, i = 0, j = 1)
    dh_x = dde.grad.jacobian(h, x, i = 0, j = 0)
    return dh_t + c * dh_x

# Da li je t=0?
def initial_h(x, on_boundary):
    return on_boundary and np.isclose(x[1], 0)

# Da li je x=0?
def boundary_hx0(x, on_boundary):
    return on_boundary and np.isclose(x[0], 0)

# Pocetni uslov za visinu vode x(t=0)
def func_init_h(x):
    return 1.751

# Dirihleov granicni uslov - Profil poplavnog talasa u vremenu
def func_hx0(x):
    t = x[:, 1:2]

    Qin = 180 * (1 + (-(np.sign(t - 600) / 2) + 0.5) * np.sin(t *  np.
↪pi / 600))
    a = Qin * n
    b = B * np.sqrt(Id)
    c = a / b
    return custom_pow(c, 3/5)

time_domain = dde.geometry.TimeDomain(0, total_time)
geom_domain = dde.geometry.Interval(0, length)
geotime = dde.geometry.GeometryXTime(geom_domain, time_domain)

# Realizacija granicnog i pocetnog uslova
bc = dde.icbc.DirichletBC(geotime, func = func_hx0, on_boundary =␣
↪boundary_hx0)
ic = dde.icbc.IC(geotime, func = func_init_h, on_initial = initial_h)

# Konstrukcija modela i definisanje kolokacionih tacaka
```

(continues on next page)

(nastavak sa prethodne stranice)

```
61 data = dde.data.TimePDE(geotime, pde, [bc, ic], num_domain = 16000,␣
   ↪num_boundary = 1000,
62       num_initial = 100, train_distribution = 'uniform')
63 net = dde.nn.FNN(layers, activation, initializer)
64 model = dde.Model(data, net)
65
66 # Treniranje RMSProp metodom
67 model.compile(optimizer = optimizer, loss = loss, lr = learning_rate)
68 loss_history, train_state = model.train(epochs = num_of_epochs,␣
   ↪display_every = 1000, batch_size = batch_size)
69
70 # Dodatno treniranje L-BFGS-B metodom posle RMSprop optimizacije
71 model.compile("L-BFGS-B")
72 loss_history, train_state = model.train()
```

U ovoj skripti odmah na početku definišemo i fizičke parametre problema i hiper-parametre modela. Pogledom na grupu hiper-parametara odmah može da se primeti značajno veći broj epoha za trening, kao i optimizator `RMSProp` umesto standardnog `Adam` optimizatora. `Adam` optimizator prilikom računanja gradijenta koristi i prvi i drugi izvod (moment), dok `RMSProp` koristi samo drugi izvod. Tokom eksperimentisanja sa različitim hiper-parametrima, ispostavilo se da za ovaj konkretan primer `RMSProp` zaista nešto brže konvergira. Takođe, pokazalo se da je primer u nekim scenarijima osetljiv čak i na izbor `batch_size` i inicijalizatora težina. Praksa je pokazala da je uz aktivacione funkcije kao što su `sigmoid` ili `tanh` bolje koristiti `Glorot` inicijalizator, dok uz aktivacionu funkciju `relu` bolje ide `He`, po Katanforoosh and Kunin [KK19].

Na žalost, oko izbora hiper-parametara ne postoje stroga pravila. Sve zavisi od samog primera, pa se izbor optimalnih hiper-parametara za neki konkretan problem uglavnom svodi na manuelnu, vremenski zahtevnu proceduru. Pomoću alata kao što je *Tensorflow/Keras* može se donekle umanjiti ovaj problem jednostavnim algoritmima kao što je nasumična pretraga (*Random Grid Search*), koja zahteva ogromne računarske resurse da bi se dobili iole upotrebljivi rezultati. S druge strane, postoji nekoliko alata koji ovu pretragu čine efikasnijom pametnijim pristupom optimizaciji. Na primer, alat Blackfox[12] koristi distribuirani genetski algoritam, a problem hardverskih resursa rešava distribuiranom obukom na lokalnom *Kubernetes* klasteru ili klasteru postavljenom na nekom klaud provajderu.

Sledi postavka početnog Dirihleovog uslova za nivo vode u kanalu i nešto složenijeg graničnog uslova za visinu vode h koji se menja u vremenu po jednačini (4.4). Ovde samo treba naglasiti da se kod DeepXDE ulazi x i t zadaju kao jedan dvokolonski tenzor, u kome je:

[12] https://blackfox.ai

```
x[:, 0:1] # ulaz x
x[:, 1:2] # ulaz t
```

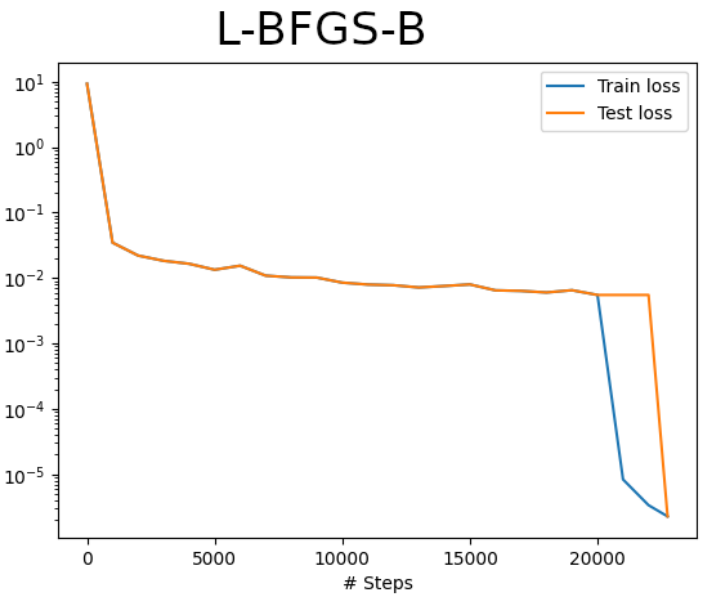
Kako je u pitanju dinamički problem koji pokriva relativno veliki prostorni i vremenski domen, tj. prati se linija od 1,6 km tokom približno 17 minuta, potreban je i veći broj kolokacionih tačaka nego u nekim problemima koje smo ranije obrađivali. Model se postavlja kao:

```
data = dde.data.TimePDE(geotime, pde, [bc, ic], num_domain = 16000,␣
↪num_boundary = 1000,
  num_initial = 100, train_distribution = 'uniform')
```

Brojnost kolokacionih tačaka za početne i granične uslove prati brojnost tačaka unutar domena. Nakon standardnog treniranja metodom `RMSProp` primećujemo još jednu specifičnost u odnosu na jednostavnije primere. Naime, nakon što se obavi „globalna" pretraga, algoritam *Limited Memory Broyden-Fletcher-Goldfarb-Shanno* ima priliku da se dodatno približi optimalnom rešenju prema Markidis [Mar21]:

```
model.compile("L-BFGS-B")
loss_history, train_state = model.train()
```

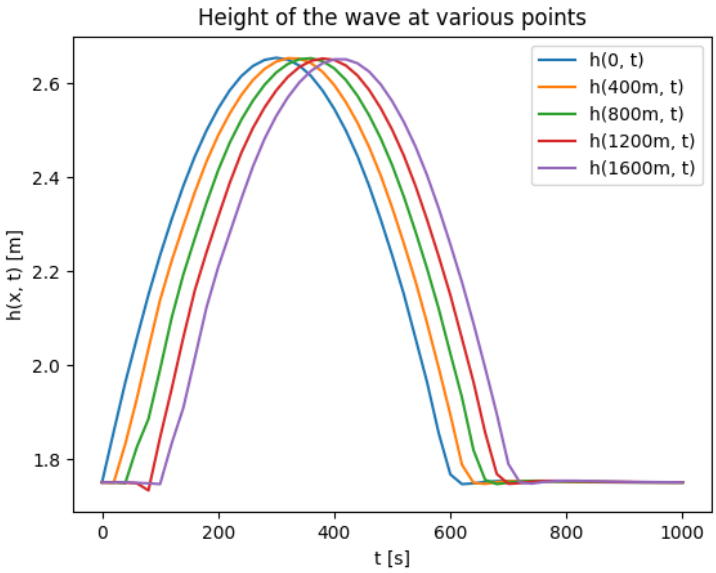
Primećujemo da se ovde ne navodi broj epoha, već se algoritam oslanja na automatsku detekciju konvergencije. Grafik funkcije gubitka se može videti na Sl. 4.9, gde do 20.000. epohe, kao što je već rečeno, teče `RMSProp`, a onda se u lokalnoj okolini nastavlja sa `L-BFGS-B`. Veoma je uočljiv rast performansi treniranja, tj. pad vrednosti funkcije gubitka u tom delu.



Sl. 4.9: Početno treniranje metodom *RMSProp* u 20.000 epoha i dodatno treniranje metodom *L-BFGS-B* do detektovane konvergencije

Što se samog procesa treniranja ove NMPFZ tiče, treba naglasiti da je za ovoliku količinu podataka, tj. kolokacionih tačaka, taj proces daleko brže radi na grafičkom procesoru nego na standardnom procesoru. Slobodna procena je da je treniranje na *Tesla T4* grafičkom procesoru više od 10 puta brže nego na procesoru *Intel Xeon Silver 4208 @ 2.10GHz*.

Konačno dolazimo i do rezultata. Visina vodenog stuba u više kontrolnih tačaka (0, 400m, 800m, 1200m, 1600m) prikazana je na Sl. 4.10. Ova rešenja se dobro poklapaju sa rešenjima koje daje metoda konačnih razlika, ali to poređenje ovde nećemo prikazivati.



Sl. 4.10: Visina vodenog talasa u nekoliko tačaka tokom vremena

4.3.4 Inverzni problem

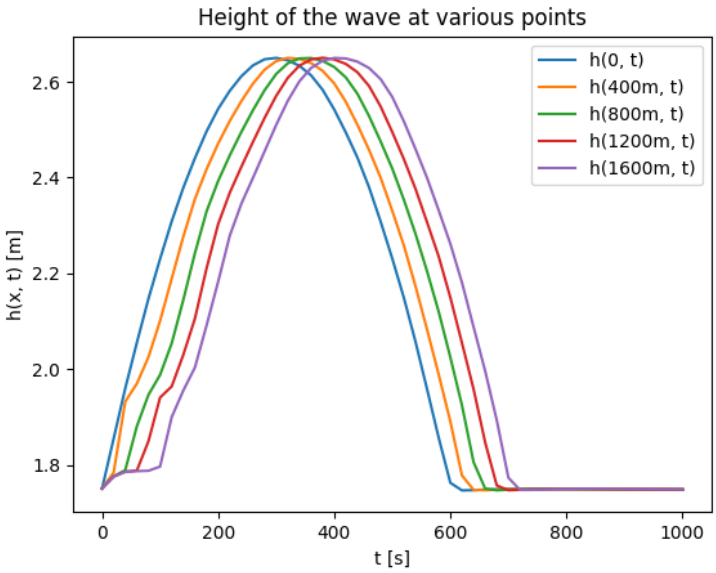
Pošto smo uspešno rešili direktni problem, hajde da zamislimo situaciju da nam nije poznat parametar n koji reprezentuje Maningovu hrapavost, ali da smo posmatranjem kretanja talasa utvrdili da je njegov vrh visine 2,65m prošao kroz kontrolne tačke (0, 400m, 800m, 1200m, 1600m) u sledećim trenucima:

x [m]	0	400	800	1200	1600
t [s]	300	320	360	380	400

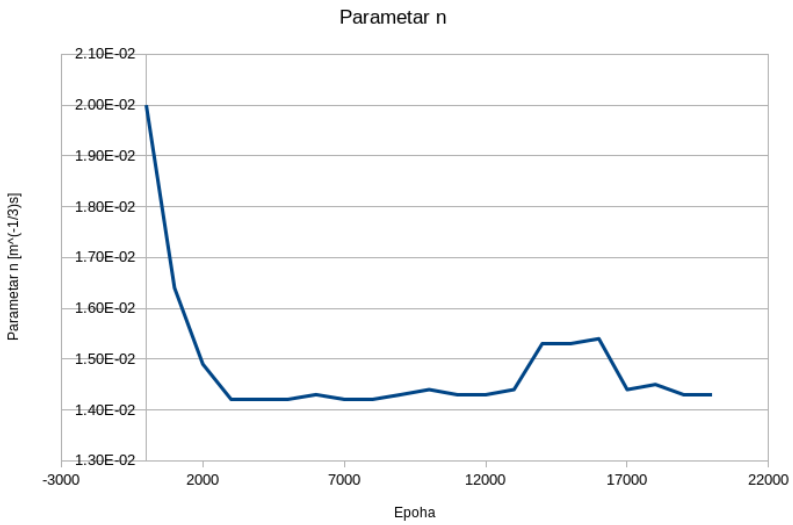
Ove opservacije čine mogućim kreiranje `PointSet` graničnog uslova koji smo već koristili, a to se kod DeepXDE okvira radi na sledeći način:

```
bc_x = np.array([[0,300],[400,320],[800,360],[1200,380],[1600,400]]).
↪reshape(5,2)
bc_y = np.array([2.65,2.65,2.65,2.65,2.65]).reshape(5,1)
ic3 = dde.icbc.PointSetBC(bc_x, bc_y, component=0)
```

Rezultati visine vodenog stuba su prikazani na Sl. 4.11, dok je vrednost parametra n tokom obuke prikazana na Sl. 4.12.



Sl. 4.11: Visina vodenog stuba kod inverznog problema



Sl. 4.12: Vrednost nepoznatog parametra n tokom obuke

Vidimo da se NMPFZ i u ovom problemu dosta dobro snalazi sa inverznom postavkom. Još jednom valja naglasiti da se kod NMPFZ direktni i inverzni pristup metodološki uopšte ne razlikuju i da zahtevaju istu količinu računarskih resursa. Nasuprot tome, klasične numeričke metode kao MKE su u stanju da reše isključivo direktne probleme. Za indentifikaciju parametara kod MKE moraju da se koriste metode za konveksnu ili češće, nekonveksnu optimizaciju koje su u realnim primenama računarski veoma zahtevne, a ponekad i nerešive.

Naravno, ni NMPFZ nije idealan. Uspešnost obuke i ovom primeru mnogo zavisi od izbora hiper-parametara, a često se dešava da usled stohastičkog karaktera same obuke ni isti hiper-parametri ne dovode do rešenja baš u svakom treningu. Pored hiper-parametara, ovde imamo i početnu vrednost nepoznatog fizičkog parametra (ili više parametara), pa se neretko dešava da optimizacija za neke početne vrednosti uopšte ne konvergira, zadržavajući se u nekom lokalnom minimumu.

5 Akustika

5.1 Uvod

U ovom poglavlju bavićemo se mogućnostima za rešavanje Helmholcove jednačine metodom NMPFZ. Helmholcova jednačina se koristi u akustici za opisivanje vibracija u nehomogenim medijumima, kao što su zvučni talasi u vazduhu ili vodi. Takođe se koristi za opisivanje različitih akustičkih pojava, kao što su rezonancija i difrakcija zvuka. Opisuje kako se zvuk širi, kako se odbija i kako reflektuje. Korisna je u projektovanju i analizi akustičkih sistema, kao što su zvučnici, mikrofoni i zvučni izolatori. Pored toga, Helmholcova jednačina se koristi i za opisivanje drugih talasnih pojava, recimo elektromagnetnih polja, u elektromagnetnoj teoriji, elektrotehnici i fizici kondenzovanog stanja, a posebno u novije vreme za projektovanje optičkih vlakana za telekomunikacije.

Ovde ćemo se ograničiti na akustiku, i to iz jednostavnog razloga. Da bi se NMPFZ ispravno obučila, potrebna je gustina kolokacionih tačaka srazmerna talasnoj dužini, tj. najmanje 10-30 tačaka po jednoj λ. Modelovanje elektromagnetnih pojava čija je karakteristična talasna dužina λ za nekoliko redova veličine manja, prevazišla bi dostupne računarske resurse, i po pitanju procesorske snage i po pitanju količine memorije. Prvi primer je elementarno prostiranje talasa u ravni. U drugom primeru naučićemo kako da postavimo malo složeniju geometriju i Nojmanove granične uslove. U prva dva primera koristićemo okvir *DeepXDE*. Treći primer uključuje i imaginarni deo funkcije, pa ćemo rešavati sistem dve parcijalne diferencijalne jednačine sa dve nepoznate funkcije. Za ovaj najkompleksniji primer biće upotrebljen okvir *SciANN*.

5.2 Rešavanje na domenu oblika kvadrata

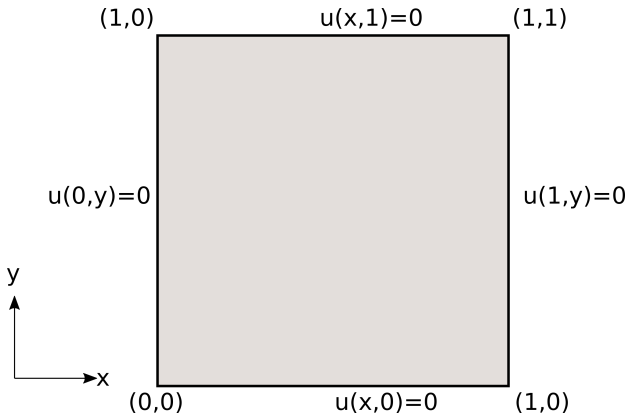
Krenućemo od najjednostavnijeg dvodimenzionog slučaja stojećeg talasa u akustici. Za talasni broj $k_0 = 2\pi n$ za $n = 2$, treba rešiti Helmholcovu (*Helmholtz*) jednačinu oblika:

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} - k_0^2 u = f, \quad \Omega = [0,1]^2,$$

uz Dirihleove granične uslove

$$u(x,y) = 0, \quad (x,y) \in \partial\Omega$$

prikazane na Sl. 5.1 i član koji specificira izvor $f(x,y) = k_0^2 \sin(k_0 x) \sin(k_0 y)$.



Sl. 5.1: Postavka problema i granični uslovi.

Postoji analitičko rešenje ovog problema i ono glasi:

$$u(x,y) = \sin(k_0 x) \sin(k_0 y).$$

Za više detalja u pogledu teorijske pozadine diferencijalne jednačine i graničnih uslova čitalac može konsultovati Ihlenburg [Ihl98]. Rešenje metodom konačnih elemenata (MKE) je takođe dostupno u okviru Dolfinx tutorijala[13].

[13] https://github.com/FEniCS/dolfinx/blob/main/python/demo/demo_helmholtz.py

5.2.1 Implementacija

Rešenje direktnog problema prikazano je na sledećem listingu.

Listing 5.1: Rešenje problema prostiranja stojećeg talasa u 2D korišćenjem DeepXDE biblioteke

```
import deepxde as dde
import numpy as np

# Frekvencija
n = 2

precision_train = 10
precision_test = 30
weights = 100
iterations = 10000
learning_rate, num_dense_layers, num_dense_nodes, activation = 1e-3, 3,
 ↪ 150, "sin"

# Uvezi sinus
from deepxde.backend import tf
sin = tf.sin

# Osnovna PDE
def pde(x, u):
    du_xx = dde.grad.hessian(u, x, i=0, j=0)
    du_yy = dde.grad.hessian(u, x, i=1, j=1)

    f = k0 ** 2 * sin(k0 * x[:, 0:1]) * sin(k0 * x[:, 1:2])
    return -du_xx - du_yy - k0 ** 2 * u - f

# Egzaktno resenje
def func(x):
    return np.sin(k0 * x[:, 0:1]) * np.sin(k0 * x[:, 1:2])

# Da li je kol. tacka na granici?
def boundary(_, on_boundary):
    return on_boundary

# Geometrija jedinicnog kvadrata
geom = dde.geometry.Rectangle([0, 0], [1, 1])
# Talasni broj
k0 = 2 * np.pi * n
# Talasna duzina
wave_len = 1 / n

```

(continues on next page)

(nastavak sa prethodne stranice)

```
hx_train = wave_len / precision_train
nx_train = int(1 / hx_train)

hx_test = wave_len / precision_test
nx_test = int(1 / hx_test)

# Dirihleov granicni uslov y=0 na granicama
bc = dde.icbc.DirichletBC(geom, lambda x: 0, boundary)

data = dde.data.PDE(
    geom,
    pde,
    bc,
    num_domain=nx_train ** 2,
    num_boundary=4 * nx_train,
    solution=func,
    num_test=nx_test ** 2,
)

# Mreza i model
net = dde.nn.FNN([2] + [num_dense_nodes] * num_dense_layers + [1], activation, "Glorot uniform")
model = dde.Model(data, net)

# Forsiraj vece tezine za granicne uslove nego za unutrasnjost domena
loss_weights = [1, weights]

model.compile("adam", lr=learning_rate, metrics=["l2 relative error"], loss_weights=loss_weights)

losshistory, train_state = model.train(iterations=iterations)
dde.saveplot(losshistory, train_state, issave=True, isplot=True)
```

Nakon standardnog importa odgovarajućih modula, počinjemo specifikacijom opštih parametara. Ovaj primer ima par specifičnosti u odnosu na ostale. Naime, da bi se uspešno modelovale talasne pojave pomoću NMPFZ, gustina kolokacionih tačaka mora da bude direktno proporcionalna frekvenciji. Što je viša frekvencija `n`, manja je talasna dužina `wave_len`, pa je potrebno više kolokacionih tačaka da pokrije domen. Ovde smo uzeli 10 kolokacionih tačaka po talasnoj dužini tokom treninga i 30 tačaka po talasnoj dužini u test skupu.

```
# Frekvencija talasa
n = 2
precision_train = 10
precision_test = 30
```

(continues on next page)

(nastavak sa prethodne stranice)

```
weights = 100
learning_rate, num_dense_layers, num_dense_nodes, activation = 1e-3, 3,
↪ 150, "sin"
```

Takođe, vidimo da koristimo arhitekturu sa manjim brojem slojeva, ali sa više neurona po sloju, kao i aktivacionu funkciju $\sin(x)$ koja bi trebalo da bude pogodnija za oponašanje talasnih fenomena.

Sledi specifikacija same parcijalne diferencijalne jednačine u obliku funkcije gubitka kako smo to već navikli:

```
def pde(x, u):
    du_xx = dde.grad.hessian(u, x, i=0, j=0)
    du_yy = dde.grad.hessian(u, x, i=1, j=1)

    f = k0 ** 2 * sin(k0 * x[:, 0:1]) * sin(k0 * x[:, 1:2])
    return -du_xx - du_yy - k0 ** 2 * u - f
```

Ovde koristimo uslužnu funkciju `dde.grad.hessian` odabirom koordinate koja se diferencira i kojom se diferencira. U ovom primeru su granični uslovi elementarni, pa ih ovde nećemo posebno navoditi.

Geometrija, talasni broj $k_0 = 2\pi\nu$ i talasna dužina $\lambda = \frac{1}{\nu}$ daju se kao:

```
geom = dde.geometry.Rectangle([0, 0], [1, 1])
k0 = 2 * np.pi * n
wave_len = 1 / n
```

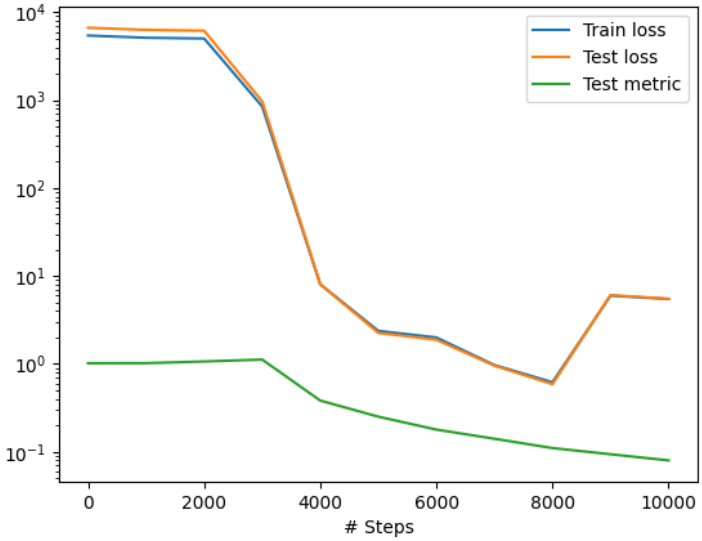
Jedina specifičnost koju dodatno treba naglasiti je da ponekad treba forsirati poštovanje graničnih uslova time što ćemo članu funkcije gubitka koji se odnosi na Dirihleov granični uslov dobiti veću težinu u odnosu na član koji se odnosi na diferencijalnu jednačinu.

```
weights = 100
loss_weights = [1, weights]
model.compile("adam", lr=learning_rate, metrics=["l2 relative error"],␣
↪loss_weights=loss_weights)
```

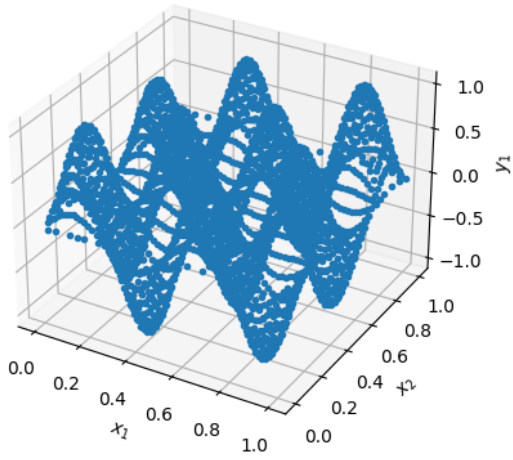
Da bi se izbegao ovaj korak koji sa sobom nosi eksperimentisanje sa različitim vrednostima težinskog faktora, granični uslov se kod *DeepXDE* može zadavati i direktnom transformacijom funkcije gubitka, ali ovde se time nećemo baviti.

5.2.2 Rezultati

Nakon 10.000 epoha obučavanja optimizacionom metodom `Adam` koji je protekao kao što je prikazano na Sl. 5.2, dobijamo stojeći talas čiji 3D prikaz možemo videti na Sl. 5.3.



Sl. 5.2: Tok obučavanja NMPFZ



Sl. 5.3: Trodimenzioni prikaz talasa u domenu oblika kvadrata

Mera greške modela RMSE (*Root Mean Squared Error*) iznosi $7,98 \cdot 10^{-2}$. Uz obraćanje posebne pažnje na forsiranje graničnih uslova, zatim arhitekturu NMPFZ i najzad tip aktivacione funkcije, uspeli smo da dobijemo prilično dobro rešenje. Čitalac može samostalno da proba kako bi promena frekvencije (a samim tim i talasne dužine), gustine kolokacionih tačaka, arhitekture, uticala na proces obučavanja modela.

Ovde možemo dati i kratku preporuku **kako pristupiti modelovanju složenijih pojava**, sa složenijom geometrijom i kompleksnijim graničnim uslovima. Pošto NMPFZ rešavanje zavisi od većeg broja hiper-parametara, preporuka je da se prvo reši do kraja pojednostavljen problem baziran na istoj diferencijalnoj jednačini, ali sa jednostavnijom geometrijom i graničnim uslovima. Kada se stekne slika o tome koja kombinacija hiper-parametara vodi do konvergencije rešenja, onda je lakše pristupiti glavnom, kompleksnom problemu. S druge strane, postoji nekoliko alata koji pretragu hiper-parametara čine efikasnijom, kao već pomenuti BlackFox[14] koji koristi distribuirani genetski algoritam.

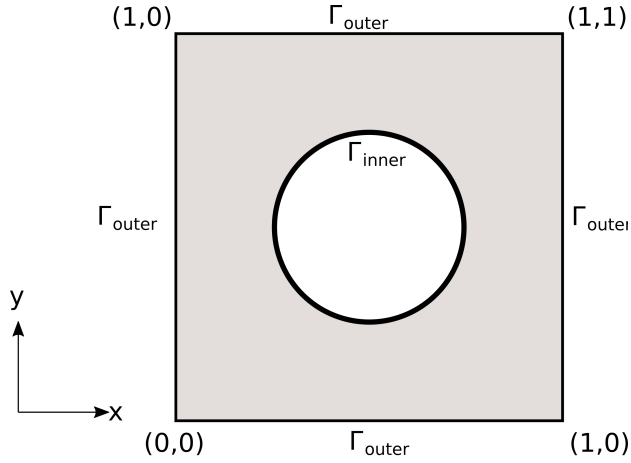
[14] https://blackfox.ai

5.3 Rešavanje na domenu oblika kvadrata sa šupljinom

U odnosu na prethodni primer *Rešavanje na domenu oblika kvadrata* (stranica 74) dodajemo šupljinu u sredini kvadratnog domena i propisujemo odgovarajuće Nojmanove granične uslove na obodu šupljine. Da se podsetimo, domen problema Ω je kvadrat stranice L, $L = 1$, iz koga isključujemo krug poluprečnika $R = \frac{1}{4}$. Za talasni broj $k_0 = 2\pi n$ i $n = 1$, rešavamo Helmholcovu jednačinu:

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} - k_0^2 u = f, \quad \text{u } \Omega,$$

gde je član koji specificira izvor $f(x, y) = k_0^2 \sin(k_0 x) \sin(k_0 y)$. Oblik domena može se videti na Sl. 5.4.



Sl. 5.4: Postavka problema i granični uslovi

Postoji analitičko rešenje ovog problema i ono glasi:

$$u(x, y) = \sin(k_0 x) \sin(k_0 y).$$

Ovde ćemo specificirati Dirihleove granične uslove prema analitičkom rešenju na spoljnoj granici domena Ω koju označavamo sa Γ_{outer}:

$$u(x, y) \mid \Gamma_{outer} = \sin(k_0 x) \sin(k_0 y), \quad (x, y) \in \Gamma_{outer}$$

Na sličan način, možemo da definišemo granični uslov na unutrašnjoj granici, ovog puta Nojmanov:

$$(\nabla u \mid \Gamma_{inner} \cdot n)(x, y) = [k_0 \cos(k_0 x) \sin(k_0 y), k_0 \sin(k_0 x) \cos(k_0 y)] \cdot n = g(x, y), \tag{5.1}$$

gde je n vektor normale. Konciznije napisano, Nojmanov granični uslov na unutrašnjoj granici glasi:

$$\nabla u \mid \Gamma_{inner} \cdot n = g.$$

5.3.1 Implementacija

Na sledećem listingu su dati glavni detalji implementacije. Namerno su izostavljeni delovi koji su irelevantni za samo rešavanje, kao što je crtanje dijagrama. Celokupna skripta se, kao i ostale, nalazi u repozitorijumu sa primerima.

Listing 5.2: Rešenje problema prostiranja stojećeg talasa u 2D domenu sa šupljinom

```
import deepxde as dde
import matplotlib.pyplot as plt
import numpy as np
from deepxde.backend import tf
sin = tf.sin

# Opsti parametri
n = 2
length = 1
R = 1 / 4

precision_train = 15
precision_test = 30

weight_inner = 10
weight_outer = 100
iterations = 5000
learning_rate = 1e-3
num_dense_layers = 3
num_dense_nodes = 350
activation = "sin"

k0 = 2 * np.pi * n
wave_len = 1 / n

# Parcijalna diferencijalna jednacina
def pde(x, y):
    dy_xx = dde.grad.hessian(y, x, i=0, j=0)
    dy_yy = dde.grad.hessian(y, x, i=1, j=1)
    f = k0**2 * sin(k0 * x[:, 0:1]) * sin(k0 * x[:, 1:2])
    return -dy_xx - dy_yy - k0**2 * y - f

```

(continues on next page)

(nastavak sa prethodne stranice)

```
# Egzaktno resenje
def func(x):
    return np.sin(k0 * x[:, 0:1]) * np.sin(k0 * x[:, 1:2])

# Da li je tacka na granici?
def boundary(_, on_boundary):
    return on_boundary

# Njumanovi granicni uslovi prema egzaktnom resenju
def neumann(x):
    grad = np.array([
            k0 * np.cos(k0 * x[:, 0:1]) * np.sin(k0 * x[:, 1:2]),
            k0 * np.sin(k0 * x[:, 0:1]) * np.cos(k0 * x[:, 1:2]),])

    normal = -inner.boundary_normal(x)
    normal = np.array([normal]).T
    result = np.sum(grad * normal, axis=0)
    return result

# Geometrija
outer = dde.geometry.Rectangle([-length / 2, -length / 2], [length / 2,
↪ length / 2])
inner = dde.geometry.Disk([0, 0], R)

# Da li je tacka na spoljnoj granici?
def boundary_outer(x, on_boundary):
    return on_boundary and outer.on_boundary(x)

# Da li je tacka na unutrasnjoj granici?
def boundary_inner(x, on_boundary):
    return on_boundary and inner.on_boundary(x)

# Iskljuci krug iz kvadrata
geom = outer - inner

hx_train = wave_len / precision_train
nx_train = int(1 / hx_train)

hx_test = wave_len / precision_test
nx_test = int(1 / hx_test)

# Na unutrasnjoj granici Njuman, na spoljnoj Dirihleovi
bc_inner = dde.icbc.NeumannBC(geom, neumann, boundary_inner)
bc_outer = dde.icbc.DirichletBC(geom, func, boundary_outer)

data = dde.data.PDE(
```

(continues on next page)

(nastavak sa prethodne stranice)

```
    geom,
    pde,
    [bc_inner, bc_outer],
    num_domain=nx_train**2,
    num_boundary=16 * nx_train,
    solution=func,
    num_test=nx_test**2,
)

net = dde.nn.FNN(
    [2] + [num_dense_nodes] * num_dense_layers + [1], activation,
↪"Glorot uniform"
)

model = dde.Model(data, net)

loss_weights = [1, weight_inner, weight_outer]
model.compile("adam", lr=learning_rate, metrics=["l2 relative error"],␣
↪loss_weights=loss_weights)

losshistory, train_state = model.train(iterations=iterations)
```

Koristićemo *Tensorflow* kao *backend* u svim našim primerima, ali treba imati u vidu da okvir *DeepXDE* podržava i *PyTorch* i još neke. Nakon standardne specifikacije opštih parametara i hiper-parametara, kao u primeru *Rešavanje na domenu oblika kvadrata* (stranica 74), uz jedinu modifikaciju dodavanja nešto više neurona po sloju (350), definišemo Nojmanov granični uslov prema jednačini (5.1):

```
def neumann(x):
    grad = np.array([
            k0 * np.cos(k0 * x[:, 0:1]) * np.sin(k0 * x[:, 1:2]),
            k0 * np.sin(k0 * x[:, 0:1]) * np.cos(k0 * x[:, 1:2]),])

    normal = -inner.boundary_normal(x)
    normal = np.array([normal]).T
    result = np.sum(grad * normal, axis=0)
    return result
```

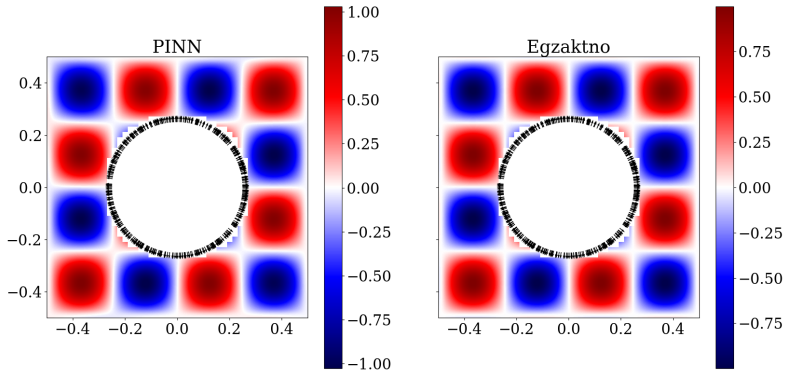
Kao što se vidi iz koda, postoje uslužne funkcije koje računaju normale na pravilne granice u kolokacionim tačkama. Ponderske težine graničnih uslova u obuci `weight_inner` i `weight_outer` takođe spadaju u neku vrstu hiper-parametara, pa i njima treba posvetiti pažnju uz nekoliko manuelnih proba. Dalje, sledi specifikacija geometrije problema kao razlike kvadrata i diska:

```
outer = dde.geometry.Rectangle([-length / 2, -length / 2], [length / 2,
↪ length / 2])
inner = dde.geometry.Disk([0, 0], R)
geom = outer - inner
```

Ostatak skripte je sličan primeru bez šupljine *Rešavanje na domenu oblika kvadrata* (stranica 74), pa ga nećemo dodatno pojašnjavati. Dovoljno je reći da pažnju treba obratiti da bude dovoljno kolokacionih tačaka na spoljnoj i na unutrašnjoj granici.

5.3.2 Rezultati

Dobijeni rezultati su prikazani u formi konturnog grafika na Sl. 5.5. Oko unutrašnje granice prikazani su pravci vektora normala.



Sl. 5.5: Rezultati primera kvadratnog domena sa šupljinom

Mera relativne greške modela iznosi 0,048. Uz obraćanje posebne pažnje na forsiranje graničnih uslova, zatim arhitekturu NMPFZ i najzad tip aktivacione funkcije, uspeli smo da dobijemo prilično dobro rešenje. Čitalac može samostalno da proba kako bi promena frekvencije (a samim tim i talasne dužine), gustine kolokacionih tačaka, arhitekture, uticala na proces obuke modela.

5.4 Dvodimenzioni problem sa sočivom

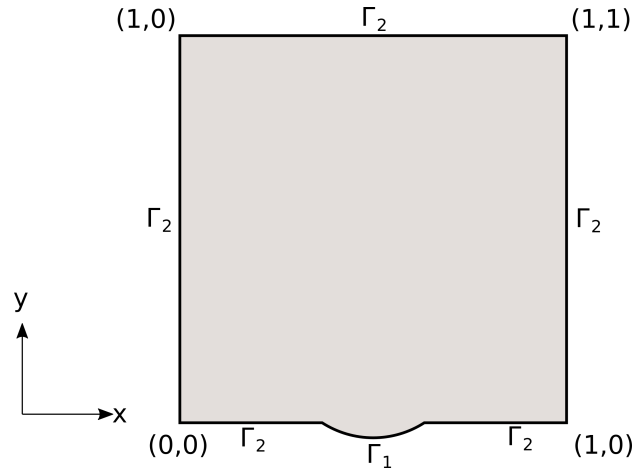
Ovaj odeljak se bavi nešto modifikovanim primerom iz tutorijala[15] za poznati softverski okvir za rad sa metodom konačnih elemenata **Deal.II**. Prvobitna svrha ovog primera je da simulira svojstva fokusiranja ultrazvučnog talasa koji generiše sočivo pretvarača sa promenljivom geometrijom. Savremene primene u medicinskom imidžingu koriste ultrazvučne talase ne samo za svrhe snimanja, već i za izazivanje određenih lokalnih efekata u materijalu, kao što su promene u optičkim svojstvima, koje se zatim mogu meriti drugim tehnikama snimanja. Vitalni sastojak ovih metoda je sposobnost fokusiranja intenziteta ultrazvučnog talasa u određenom delu materijala, idealno u tački, kako bi se mogla ispitati svojstva materijala na toj lokaciji.

Karakteristična talasna dužina ultrazvuka nešto manja od fenomena koje smo do sada modelovali metodom NMPFZ, te bi nam bio potreban izuzetno veliki broj kolokacionih tačaka što produžava treniranje. Zato smo talasnu dužinu nešto povećali (smanjili frekvenciju), dok ostale parametre nismo menjali.

Kako bismo izveli jednačine za ovaj problem, zvuk uzimamo kao talas kojim se širi promena pritiska:

$$\frac{\partial^2 U}{\partial t^2} - c^2 \Delta U = 0,$$

gde je c brzina zvuka, koja se zbog jednostavnosti uzima kao konstanta, $U = U(x,t),\ x \in \Omega,\ t \in \mathrm{R}$. Postavka problema data je na Sl. 5.6.



Sl. 5.6: Postavka problema i granični uslovi

[15] https://www.dealii.org/current/doxygen/deal.II/step_29.html

Granica $\Gamma = \partial\Omega$ podeljena je na dva dela, i to Γ_1 i $\Gamma_2 = \Gamma \setminus \Gamma_1$, gde Γ_1 predstavlja sočivo, a Γ_2 apsorbujuću granicu. Zapravo, želimo da napravimo takav granični uslov na Γ_2 tako da se oponaša znatno veći domen. Na Γ_1, pretvarač generiše talase konstantne frekvencije $\omega > 0$ i konstantne jedinične amplitude:

$$U(x,t) = \cos\omega t, \qquad x \in \Gamma_1$$

Pošto nema drugih (internih ili graničnih) izvora i pošto samo izvor emituje talase frekvencije ω, dozvoljeno je da izvršimo razdvajanje promenljivih $U(x,t) = \mathrm{Re}\left(u(x)\, e^{i\omega t}\right)$. Kompleksna funkcija $u(x)$ opisuje prostornu zavisnost amplitude i faze (relativno u odnosu na izvor) talasa frekvencije ω, dok je amplituda veličina koja nas interesuje. Ako ovako formulisanu funkciju uvrstimo u talasnu jednačinu, vidimo da za u imamo

$$\begin{aligned} -\omega^2 u(x) - c^2 \Delta u(x) = 0, &\qquad x \in \Omega, \\ u(x) = 1, &\qquad x \in \Gamma_1. \end{aligned}$$

Da bismo našli odgovarajuće granične uslove na Γ_2 koji oponašaju apsorbujuću granicu, razmotrimo talas oblika $V(x,t) = e^{i(k\cdot x - \omega t)}$ frekvencije ω koji se prostire u pravcu $k \in \mathrm{R}^2$. Da bi V bio rešenje talasne jednačine, mora da važi $|k| = \frac{\omega}{c}$. Pretpostavimo da talas dolazi do $x_0 \in \Gamma_2$ pod pravim uglom, na primer $n = \frac{k}{|k|}$ gde n označava normalu na $\Omega \in x_0$. Onda u x_0, ovaj talas zadovoljava jednačinu

$$c(n \cdot \nabla V) + \frac{\partial V}{\partial t} = (i\, c\, |k| - i\, \omega) V = 0.$$

Postavljanjem graničnog uslova

$$c(n \cdot \nabla U) + \frac{\partial U}{\partial t} = 0, \qquad x \in \Gamma_2,$$

talasi koji udaraju u granicu Γ_2 pod pravim uglom biće savršeno apsorbovani. Sa druge strane, oni delovi talasnog polja koji ne padaju pod pravim uglom ne zadovoljavaju ovaj uslov, pa će dolaziti do parcijalnih refleksija. U osnovi, direktni delovi talasa će proći kroz granicu kao da ona ne postoji, dok će ostali biti reflektovani nazad u domen.

Ukoliko smo spremni da prihvatimo ovako predloženu aproksimaciju, onda za u važi sledeće:

$$\begin{aligned} -\omega^2 u - c^2 \Delta u = 0, &\qquad x \in \Omega, \\ c(n \cdot \nabla u) + i\, \omega\, u = 0, &\qquad x \in \Gamma_2, \\ u = 1, &\qquad x \in \Gamma_1. \end{aligned} \tag{5.2}$$

prepoznajemo Helmholcovu jednačinu sa Dirihleovim uslovom na Γ_1 i mešanim graničnim uslovom na Γ_2. Zbog uslova na Γ_2 ne možemo da tretiramo realne i imaginarne

delove u posebno. Ono što možemo da uradimo je da formiramo sistem od dve parcijalne diferencijalne jednačine u kojima figurišu realni i imaginarni deo u, sa graničnim uslovima na Γ_2 koje vezuju ove dve komponente. Ako označimo da je $v = \text{Re } u,\ w = \text{Im } u$, sistem (5.2) glasi:

$$\begin{aligned} -\omega^2 v - c^2 \Delta v &= 0 \\ -\omega^2 w - c^2 \Delta w &= 0 \\ x &\in \Omega, \\ c(n \cdot \nabla v) - \omega\, w &= 0 \\ c(n \cdot \nabla w) + \omega\, v &= 0 \\ x &\in \Gamma_2, \\ v &= 1 \\ w &= 0 \\ x &\in \Gamma_1. \end{aligned} \tag{5.3}$$

Dakle, prve dve jednačine važe u celom domenu Ω, druge dve na granici Γ_2, a poslednje dve na Γ_1. Ovde prvi put imamo sistem diferencijalnih jednačina, ali ni to ne bi trebalo da bude problem za NMPFZ pristup, ako podrazumevamo da je sistem zatvoren, tj. jednoznačan.

5.4.1 Implementacija

Na osnovu sistema jednačina (5.3) treba da formiramo kompozitnu funkciju gubitka, da formiramo NMPFZ mrežu i da je istreniramo na dovoljnom broju kolokacionih tačaka. Evo ključnih delova implementacije ostvarene pomoću okvira SCIANN:

Listing 5.3: Rešenje problema prostiranja talasa u 2D domenu sa sočivom

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  import sciann as sn
4  from numpy import pi
5  from sciann.utils.math import diff, sign, sin, sqrt
6
7  # Brzina talasa
8  c = 1
9  # Frekvencija
10 omega = 2*pi*4
11
12 x = sn.Variable('x')
```

(continues on next page)

(nastavak sa prethodne stranice)

```
y = sn.Variable('y')
v, w = sn.Functional (["v", "w"], [x, y], 3*[200] , 'sin')

# Diferencijalne jednacine za v i w
L1 = -omega**2 * v - c**2 * diff(v, x, order=2) - c**2 * diff(v, y, order=2)
L2 = -omega**2 * w - c**2 * diff(w, x, order=2) - c**2 * diff(w, y, order=2)

TOL = 0.015

# Dirihleov uslov na G1 (y=0 i 0.4<x<0.6)
a,b,c,d =  0.39762422, -1.57715550, -0.03696364,  1.60337246
C1 = (1 - sign(y - (a + b*x + c*sqrt(x) + d*x**2 + TOL))) * (1 + sign(x-0.4)) * (1 - sign(x-0.6)) * (1-v)
C2 = (1 - sign(y - (a + b*x + c*sqrt(x) + d*x**2 + TOL))) * (1 + sign(x-0.4)) * (1 - sign(x-0.6)) * (w-0)

# Gornja granica G2 (gde je y=1)
C3 =  (1+sign(y - (1-TOL))) * ( c*diff(v,y) - omega*w )
C4 =  (1+sign(y - (1-TOL))) * ( c*diff(w,y) + omega*v )

# Desna granica G2 (gde je x=1)
C5 =  (1+sign(x - (1-TOL))) * ( c*diff(v,x) - omega*w )
C6 =  (1+sign(x - (1-TOL))) * ( c*diff(w,x) + omega*v )

# Leva granica G2 (gde je x=0)
C7 =  (1-sign(x - (0+TOL))) * ( -c*diff(v,x) - omega*w )
C8 =  (1-sign(x - (0+TOL))) * ( -c*diff(w,x) + omega*v )

# Donja granica G2 (gde je y=0) i (x<0.4 or x>0.6)
C9 =   (1-sign(y - (0+TOL))) * ( (1 - sign(x-0.4)) + (1 + sign(x-0.6)) ) * ( -c*diff(v,y) - omega*w )
C10 =  (1-sign(y - (0+TOL))) * ( (1 - sign(x-0.4)) + (1 + sign(x-0.6)) ) * ( -c*diff(w,y) + omega*v )

x_data, y_data = [], []

kolokacione_tacke = np.genfromtxt('kolokacione_tacke.txt', delimiter=",")

for e in kolokacione_tacke:
    ind, x1, y1 = e
    x_data.append(x1)
    y_data.append(y1)

```

(continues on next page)

(nastavak sa prethodne stranice)

```
x_data, y_data = np.array(x_data), np.array(y_data)

# Model i obucavanje
m = sn.SciModel([x, y], [L1,L2,C1,C2,C3,C4,C5,C6,C7,C8,C9,C10], 'mse',
 ↪'Adam')
h = m.train([x_data, y_data], 12*['zero'], learning_rate=0.001, batch_
 ↪size=1024, epochs=8000, adaptive_weights={'method':'NTK', 'freq':200}
 ↪)

# Test
x_test, y_test = np.meshgrid(
    np.linspace(0, 1, 200),
    np.linspace(0, 1, 200)
)
v_pred = v.eval(m, [x_test, y_test])
w_pred = w.eval(m, [x_test, y_test])
```

Nakon uobičajenih importa paketa, formiramo NMPFZ mreže za realni deo v i imaginarni deo w nepoznate funkcije u. I ovde ćemo ići sa aktivacionom funkcijom $\sin(x)$. Interesantan deo koda je definisanje graničnog uslova na Γ_1 prema poslednje dve jednačine u sistemu (5.3):

```
a,b,c =  0.39762422, -1.57715550, 1.60337246
C1 = (1 - sign(y - (a + b*x + c*x**2 + TOL))) * (1 + sign(x-0.4)) * (1␣
 ↪- sign(x-0.6)) * (1-v)
C2 = (1 - sign(y - (a + b*x + c*x**2 + TOL))) * (1 + sign(x-0.4)) * (1␣
 ↪- sign(x-0.6)) * (w-0)
```

Jednačina $y = a + b \cdot x + c \cdot x^2$ predstavlja jednačinu spoljne linije sočiva Γ_1, u kojoj su koeficijenti a, b, c dobijeni fitovanjem. Dakle, prva zagrada u graničnim uslovima znači da uzimamo kolokacione tačke koje pripadaju tankom pojasu iznad linije Γ_1, dok druga i treća zagrada imaju nenultu vrednost samo ako je $0,4 < x < 0,6$.

Granični uslovi `C3, C4, C5, C6, C7, C8` se odnose na mešanu formulaciju prema druge dve jednačine u sistemu (5.3) i važe na Γ_2. Na primer:

```
C5 =  (1+sign(x - (1-TOL))) * ( c*diff(v,x) - omega*w )
C6 =  (1+sign(x - (1-TOL))) * ( c*diff(w,x) + omega*v )
```

se odnosi na desnu granicu gde je `x=1` i uzima kolokacione tačke koje se nalaze u tankom pojasu širine `TOL` sa leve strane te granice.

Komponente funkcije gubitka `C9` i `C10` odnose se takođe na granicu Γ_2, ali na liniji gde je $y = 0$` i $x < 0,4$ ili $x > 0,6$:

```
C9 =   (1-sign(y - (0+TOL))) * ( (1 - sign(x-0.4)) + (1 + sign(x-0.6))␣
↪) * ( -c*diff(v,y) - omega*w )
C10 =  (1-sign(y - (0+TOL))) * ( (1 - sign(x-0.4)) + (1 + sign(x-0.6))␣
↪) * ( -c*diff(w,y) + omega*v )
```

Ovim smo kompletirali svih 12 komponenti funkcije gubitka. Pošto ih ima toliko, nije jednostavno izvršiti njihovo ponderisanje, odnosno dodelu težina svakoj komponenti. U ovakvim situacijama pomažu metode za adaptivno određivanje težina komponenti tokom obuke. U našem rešenju:

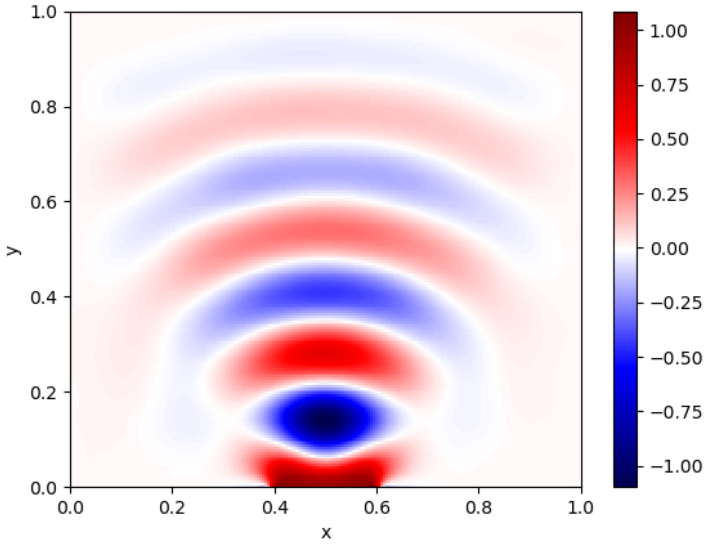
```
h = m.train([x_data, y_data], 12*['zero'], learning_rate=0.001, batch_
↪size=1024, epochs=8000, adaptive_weights={'method':'NTK', 'freq':200}
↪)
```

upotrebili smo inovativnu metodu *Neural Tangent Kernel* (NTK) prema Wang *et al.* [WYP22]. Objašnjenje metode izlazi iz okvira ovog praktikuma, pa je nećemo detaljno razrađivati. Takođe, valja napomenuti da smo kolokacione tačke učitali iz posebnog fajla `kolokacione_tacke.txt`, koji je dobijen tako što smo ispisali čvorove konačnih elemenata koji se dobijaju iz generatora mreže paketa Deal.II[16].

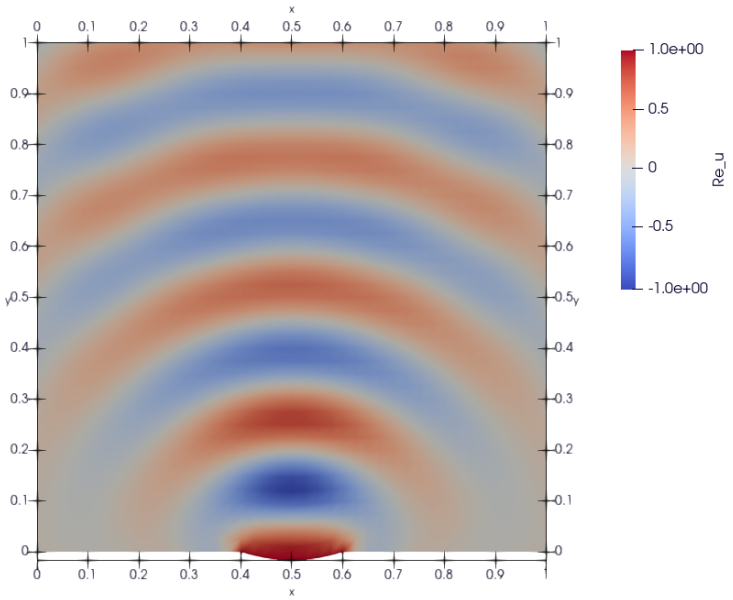
5.4.2 Rezultati

Kao što je na početku odeljka već rečeno, primer je preuzet iz dokumentacije za paket koji se bavi analizom metodom konačnih elemenata Deal.II[17], tako da možemo da uporedimo rešenje za $v = \text{Re}\ u$ koje smo dobili pomoću NMPFZ (Sl. 5.7) i rešenje koje se dobija klasičnom metodom konačnih elemenata (Sl. 5.8). Rešenje dobijeno MKE metodom možemo smatrati referentnim, jer je korišćena veoma gusta mreža i pokazana je konvergencija.

[16] https://www.dealii.org
[17] https://www.dealii.org/current/doxygen/deal.II/step_29.html



Sl. 5.7: Rešenje za $v = \text{Re } u$ dobijeno pomoću NMPFZ



Sl. 5.8: Rešenje za $v = \text{Re } u$ dobijeno metodom konačnih elemenata

Kvalitativno gledano, NMPFZ rešenje ima iste karakteristike kao MKE rešenje. Međutim, oko sočiva i prima granicama Γ_2 očigledan je pad kvaliteta rešenja. Uzrok možemo tražiti na nekoliko mesta:

- Prvo, kod MKE je izvor na liniji sočiva Γ_1 moguće preciznije specificirati po samoj liniji, a ne u pojedinačnim tačkama kao kod NMPFZ.
- Moguće je da 50.000 kolokacionih tačaka nije dovoljno za obučavanje.
- Primećeno je da obučavanje optimizacionim algoritam `Adam` ne može da spusti vrednost gubitka ispod neke granice. Ovde verovatno treba eksperimentisati sa varijabilnom stopom učenja, ili dodati neki drugi optimizator kao u primeru *Propagacija talasa u otvorenom kanalu* (stranica 62).

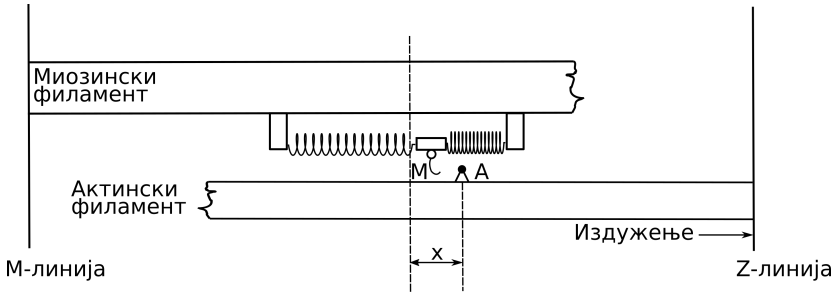
Dalje eksperimentisanje na ovom primeru ostavljamo čitaocu.

6 Modelovanje mišića

6.1 Uvod

Haksli i Hodžkin su 1957. godine razvili biofizički model koji je bio osnova za sva naredna istraživanja u ovoj oblasti Huxley [Hux57]. Priroda Hakslijevog modela se oslanja na dinamiku poprečnih mostova, zbog čega je sama teorija i nazvana teorija poprečnih mostova Huxley [Hux57]. Detalje vezane za anatomiju i fiziologiju mišića ne možemo izlagati u ovom tipu literature, već se čitalac upućuje na Svičević [Svivcevic20].

Haksli je razmatrao polovinu sarkomere i smatrao je da su glave miozina za miozinski filament pričvršćene pomoću elastičnih veza. Na Sl. 6.1 imamo šematsku reprezentaciju modela, na kojoj se može uočiti da je debeli miozinski filament fiksiran u prostoru za M-liniju. Prilikom stimulacije mišića očekivano je da se miozinske glave vežu na najbliže slobodno mesto na aktinu, usled čega dolazi do formiranja elastičnih veza u vidu poprečnih mostova. U tom trenutku dolazi do generisanja aktivne mišićne sile.



Sl. 6.1: Hakslijev modela klizajućih filamenata

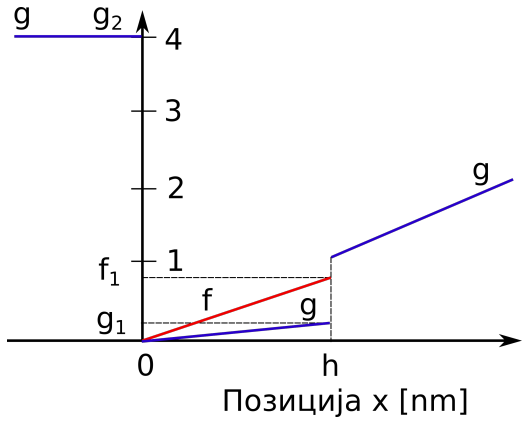
Sila se tada prenosi na aktinski filament koji se tom prilikom kreće ka Z-liniji. Proces formiranja poprečnih mostova se ponavlja usled stalnog relativnog klizanja aktinskog filamenta duž miozinskog i zavisi od položaja samih miozinskih glava. Tom prilikom, poprečni mostovi trpe istezanja i skraćivanja. Zbog stalnog relativnog klizanja filamenata, Hakslijeva teorija se naziva i Teorijom klizajućih filamenata Huxley [Hux57].

U slučaju izduženja mišića, aktinski filament klizi udesno, duž miozinskog filamenta, koji je fiksiran za M-liniju. Pomeranje glave miozina od njene uspravne pozicije, u pravcu M--linije ili Z-linije do aktivnog mesta *A* na aktinskom filamentu, označeno je sa x i tumači se kao dužina poprečnog mosta. U jednom trenutku, miozinska glava može biti vezana samo za jedno aktivno mesto aktina, pri čemu je pomeranje ograničeno maksimalnim pomeranjem glave miozina, izazvanim termičkim fluktuacijama, h , tako da važi $0 < x < h$.Ukoliko dužina poprečnog mosta postane veća od h , dolazi do raskidanja ove veze između filamenata. U svakom trenutku je moguće identifikovati da li je neka miozinska glava zakačena ili ne, i da li tom prilikom formira poprečni most dužine x. Verovatnoća da slučajno izabrana miozinska glava u trenutku t formira poprečni most dužine x iz domena Ω , označena je sa $n(x,t)$.

Verovatnoća $n(x,t)$ se može tumačiti i kao udeo broja miozinskih glava koje su u trenutku t zakačene na rastojanju x u odnosu na ukupan broj miozinskih glava. Ovaj broj zavisi od brzine uspostavljanja i raskidanja poprečnih mostova, tako da se kontinuiran proces stvaranja i raskidanja poprečnih mostova može formulisati jednačinom:

$$\frac{dn(x,t)}{dt} = [1 - n(x,t)]\, f(x) - n(x,t)g(x), \tag{6.1}$$

gde su $f(x)$ i $g(x)$ redom stope uspostavljanja i raskidanja veze između miozina i aktina u jedinici vremena, koje zavise od rastojanja x, kao na Sl. 6.2. Verovatnoća uspostavljanja veze je predstavljena proizvodom udela onih miozinskih glava koje još uvek nisu zakačene, $1 - n(x,t)$, i stope uspostavljanja veze, $f(x)$. S druge strane, verovatnoća da se uspostavljena veza između aktina i miozina prekine je data kao $n(x,t)g(x)$.



Sl. 6.2: Stope uspostavljanja veze, f, (narandžasta linija) i raskidanja veze, g, (plava linija) između miozina i aktina

Hakslijeva teorija kinetike poprečnih mostova se može izraziti korišćenjem parcijalne diferencijalne jednačine nad domenom Ω:

$$\frac{\partial n}{\partial t}(x,t) - v\frac{\partial n}{\partial x} = \mathcal{N}\left(n(x,t),x\right), \qquad \forall x \in \Omega,$$

gde je $v = -dx/dt$ brzina klizanja filamenta aktina u odnosu na filament miozina (pozitivna pri kontrakciji), a

$$\mathcal{N}\left(n(x,t),x\right) = \left[1 - n(x,t)\right] f(x) - n(x,t)g(x)$$

predstavlja brzinu promene stanja poprečnih mostova.

U cilju što realističnijeg opisa ponašanja mišića tokom izduženja, Zahalak je uveo određene modifikacije originalnog Hakslijevog modela. Uveo je minimalne promene u definiciji stopa uspostavljanja i raskidanja veza između miozina i aktina, tako da je pri skraćivanju mišića sve ostalo nepromenjeno, dok je pri izduženju omogućio veću stopu otkačinjanja. Ovo je praktično realizovano tako što je uveden Zahalakov faktor, f_{Zah}, u slučaju kada je dužina poprečnog mosta x veća od h:

$$\begin{aligned} x < 0 \quad &: \quad f(x) = 0;\ g(x) = g_2; \\ 0 \le x \le h \quad &: \quad f(x) = f_1 x/h;\ g(x) = g_1 x/h; \\ x > h \quad &: \quad f(x) = 0;\ g(x) = f_{Zah}\, g_1 x/h \end{aligned} \tag{6.2}$$

Može se uočiti da je u oblasti $x < 0$ definisana visoka vrednost stope otkačinjanja, $g(x) = g_2$, kako bi se poprečni mostovi koji su dospeli u ovu zonu brzo prekinuli. Postoji izvesna stopa otkačinjanja i u oblasti $0 < x < h$, ali je niska u poređenju sa negativnom oblašću.

Nakon završene obuke neuronske mreže, ona se može koristiti kao zamena za **metod karakteristika** objašnjen u Svičević [Svivcevic20]. Ovde ćemo se zaustaviti sa objašnjavanjem matematičkog modela, a čitaoca koji se interesuje za izučavanje ove oblasti uputiti na Svičević [Svivcevic20].

6.2 Izometrijski slučaj

Radi testiranja sposobnosti NMPFZ da rešavaju ovu vrstu parcijalnih diferencijalnih jednačina, najpre je od interesa izometrijski slučaj, jer je on najjednostavniji, budući da je **brzina klizanja filamenata jednaka nuli**. Kod izometrijskog slučaja, višeslojni perceptron kao ulaz uzima poziciju x dostupnog aktinskig sajta u odnosu na ravnotežni položaj miozinske glave i vreme t, a predviđa verovatnoće zakačinjanja miozinskih glava za aktinske sajtove $n(x,t)$.

6.2.1 Implementacija

Kompletan kod je dat na sledećem listingu.

Listing 6.1: Rešenje Hakslijeve jednačine za izometrijski slučaj

```
import deepxde as dde
import numpy as np
import tensorflow as tf

''' fixed parameters '''
f1_0 = 43.3
h = 15.6
g1 = 10.0
g2 = 209.0
fzah = 4.0
a = 1.

# Verovatnoca kacenja
def f(x,a):
    return (1+tf.sign(x)) * (1-tf.sign(x-h)) * (f1_0*a*x/h) * 0.25

# Verovatnoca raskacivanja
def g(x):
    return 0.5 * (1-tf.sign(x)) * g2 + \
        0.25 * (1+tf.sign(x)) * (1-tf.sign(x-h)) * (g1*x/h) + \
        0.5 * (1+tf.sign(x-h)) * (fzah*g1*x/h)
```

(continues on next page)

(nastavak sa prethodne stranice)

```
22
23  # n = n(x,t)
24  def pde(x, n):
25      dn_dt = dde.grad.jacobian(n, x, i=0, j=1)
26      loss = dn_dt - (1.0-n) * f(x[:,0:1],a) + n*g(x[:,0:1])
27      # Obezbedi pozitivna resenja
28      return loss + n*(1-tf.sign(n))
29
30  # Computational geometry
31  geom = dde.geometry.Interval(-20.8, 63)
32  timedomain = dde.geometry.TimeDomain(0, 0.4)
33  geomtime = dde.geometry.GeometryXTime(geom, timedomain)
34
35  # Pocetni uslovi
36  ic1 = dde.icbc.IC(geomtime, lambda x: 0.0, lambda _, on_initial: on_
    ↪initial)
37
38  data = dde.data.TimePDE(geomtime, pde, [ic1], num_domain=10000, num_
    ↪boundary=100, num_initial=500)
39  net = dde.nn.FNN([2] + [40] * 3 + [1], "tanh", "Glorot normal")
40  model = dde.Model(data, net)
41
42  model.compile("adam", lr=1e-3)
43  model.train(100000)
44  model.compile("L-BFGS", loss_weights=[1.e-1, 1])
45  losshistory, train_state = model.train()
46  dde.saveplot(losshistory, train_state, issave=True, isplot=True)
```

Konstruisana je mreža sa 3 sloja, sa po 40 neurona i aktivacionom funkcijom `tanh`. Generisana je mreža podataka za 10.000 ekvidistantnih vrednosti za x u opsegu $-20.8[nm] \leq x \leq 63\,[nm]$ i sa 500 nasumičnih vrednosti za t u opsegu $0\,[s] \leq t \leq 0.4\,[s]$. Ove generisane tačke su korišćene kao ulazni podaci za obuku mreže. U toku obuke, od 100.000 epoha, minimizuje se rezidual Hakslijeve jednačine za mišićnu kontrakciju i rezidual početnog uslova, korišćenjem Adam optimizacije sa stopom učenja 10^{-3}.

Što se same implementacije tiče, treba skrenuti pažnju na nekoliko bitnih momenata. Kao prvo, da bismo formulisali jednačine (6.2) ne savetuje se korišćenje uslovih izraza (`if` klauzula), već se ista funkcionalnost interpretira pomoću *TensorFlow* izraza za znak, ako se on koristi kao bekend. Na primer:

```
def g(x):
    return 0.5 * (1-tf.sign(x)) * g2 + \
        0.25 * (1+tf.sign(x)) * (1-tf.sign(x-h)) * (g1*x/h) + \
        0.5 * (1+tf.sign(x-h)) * (fzah*g1*x/h)
```

U funkciji parcijalne diferencijalne jednačine imamo nešto drugačiji problem. Naime, moramo da obezbedimo da funkcija $n(x,t)$ ne počne da uzima vrednosti manje od nule. Iako sama matematička postavka to dozvoljava, podsetimo se da $n(x,t)$ označava verovatnoću zakačinjanja, pa negativna vrednost nema nikakvog fizičkog smisla. To ćemo uraditi na sledeći način:

```
def pde(x, n):
    dn_dt = dde.grad.jacobian(n, x, i=0, j=1)
    loss = dn_dt - (1.0-n) * f(x[:,0:1],a) + n*g(x[:,0:1])
    # Obezbedi pozitivna resenja
    return loss + n*(1-tf.sign(n))
```

U poslednjoj liniji se vrednosti funkcije gubitka dodaje član koji ima nenultu vrednost u slučaju da je n negativno. Na prvi pogled, bilo bi dovoljno umesto `n*(1-tf.sign(n))` postaviti samo `1-tf.sign(n)`. Međutim, ispostavlja se da tako formirana funkcija gubitka ne dovodi do konvergencije, jer nije diferencijabilna, pa optimizacioni algoritmi kao što je `Adam` ne konvergiraju. Kada se pak funkcija znaka pomnoži sa n, dobijamo diferencijabilniju funkciju gubitka, a samim tim i veću verovatnoću konvergencije.

Kao i u primeru *Propagacija talasa u otvorenom kanalu* (stranica 62), i ovde ćemo pokušati da dodatno smanjimo vrednost funkcije gubitka metodom L-BFGS:

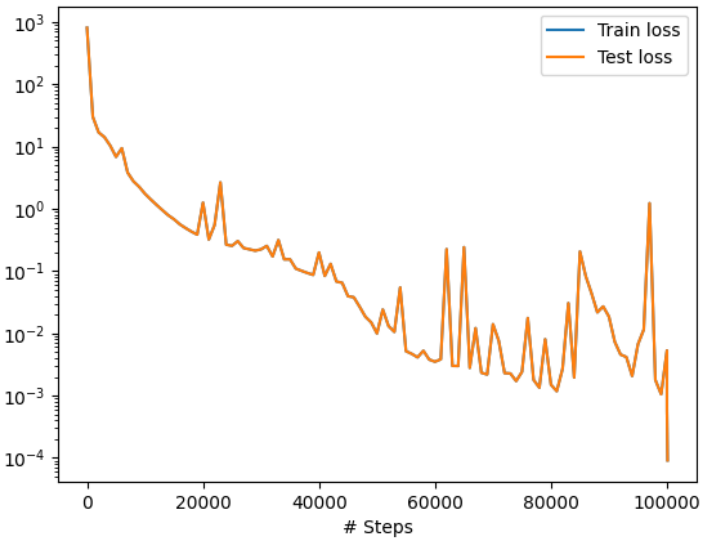
```
model.compile("L-BFGS", loss_weights=[1.e-1, 1])
losshistory, train_state = model.train()
```

Veoma je važno ispoštovati postavljene početne uslove, pa ćemo im pomoću `loss_weights=[1.e-1, 1]` dati za red veličine veću težinu od komponente koja sledi iz same parcijalne diferencijalne jednačine.

6.2.2 Rezultati

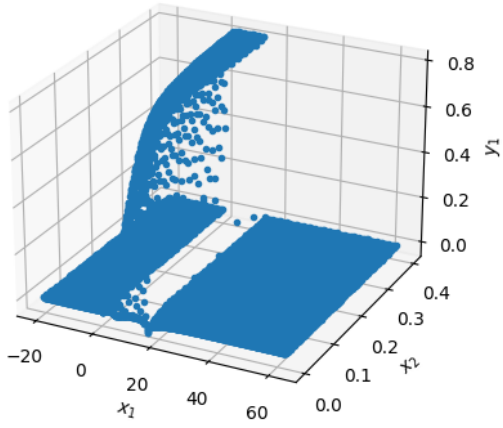
I ovaj primer će usled velikog broja kolokacionih tačaka (podataka) najbolje performanse imati ukoliko se izvršava na nekom grafičkom procesoru koji ovu masovnost ume da paralelizuje. Na primer, na našem grafičkom procesoru NVidia Tesla T4 obuka traje oko 170 sekundi, dok na osmojezgarnom procesoru Intel Xeon Silver 4208 na 2.10GHz traje 2150 sekundi. To je ubrzanje od gotovo 13 puta!

Na Sl. 6.3 se vidi kako je tekao proces treninga NMPFZ. Uočljivo je da dodatno obučavanje pomoću L-BFGS definitivno ima efekta i da smo pomoću njega spustili vrednost gubitka za dodatni red veličine.



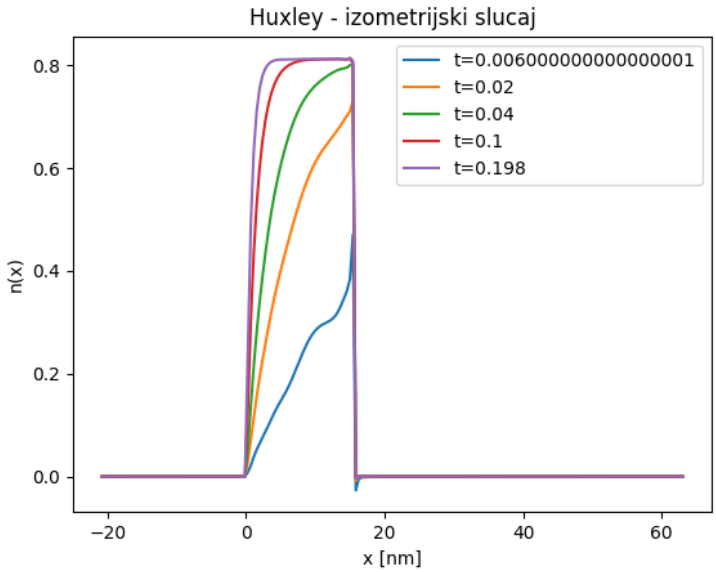
Sl. 6.3: Funkcija gubitka tokom obučavanja

Trodimenzionalni prikaz rezultata dat je na Sl. 6.4. Iz ovakvog prikaza nije pogodno utvrđivati bilo kakvu tačnost rešenja, ali je dovoljan za kvalitativni uvid. Uočljivo je da blizu $t = 0$ (x_2 na slici) ima nekoliko kolokacionih tačaka gde je $n(x, t) < 0$, ali sveukupno rešenje deluje logično.



Sl. 6.4: Trodimenzionalni prikaz dobijenih rezultata. x_1 je prostorna koordinata, a x_2 vremenska.

Precizniju vizuelnu analizu rezultata možemo obaviti tek grafičkim predstavljanjem verovatnoće zakačinjanja mostova $n(x,t)$ duž x ose u nekoliko različitih vremenskih trenutaka, kao što je dato na Sl. 6.5. Kao što smo očekivali, uočljiva je mala nestabilnost oko tačke $x = h$ u nekoliko prvih vremenskih koraka, ali se ona gubi kako proces zakačinjanja napreduje.



Sl. 6.5: Veličina $n(x,t)$ u nekoliko različitih vremenskih trenutaka

Upoređivanje rezultata sa onima koji su dobijeni metodom karakteristika izlazi iz okvira ovog praktikuma, pa ćemo taj deo preskočiti. Na kraju je važno napomenuti da jednu potencijalnu primenu NMPFZ na koju do sada nismo obraćali pažnju, a može biti od velike koristi. Naime, dok kod klasičnih numeričkih metoda za rešavanje parcijalnih diferencijalnih jednačina, rešenje u vremenskom koraku $m+1$ zavisi od rešenja koje smo imali u vremenskom koraku m. Kod svih primera koje smo obradili pomoću NMPFZ to nije slučaj, jer se vreme uzima kao bilo koja druga promenljiva po kojoj se vrši parcijalna diferencijacija. Ako pogledamo Sl. 6.5 očigledno je da u trenutku $t = 0,006$ imamo grešku. Međutim, ta **greška se ne propagira na kasnije vremenske trenutke** baš iz navedenog razloga.

Ovaj drugačiji tretman vremena kao ulazne promenljive ima implikacije i na performanse. Naime, kada se NMPFZ model koristi u produkciji, i potrebno nam je da znamo npr. $n(x, t = t_1)$ nije potrebno da prođemo kroz sve vremenske korake $t \leq t_1$, već odmah možemo da izbacimo rezultat, jednim prolaskom kroz obučenu NMPFZ. Time se vreme značajno štedi i za nekoliko redova veličine. Time se otvaraju neke nove primene, naročito u oblasti modelovanja na više skala, kao na primer u Svičević [Svivcevic20]. Takvim skokom u performansama bilo bi moguće ovakve složene modele izvoditi skoro u realnom vremenu.

7 Modelovanje proizvodnje solarnih elektrana

7.1 Uvod

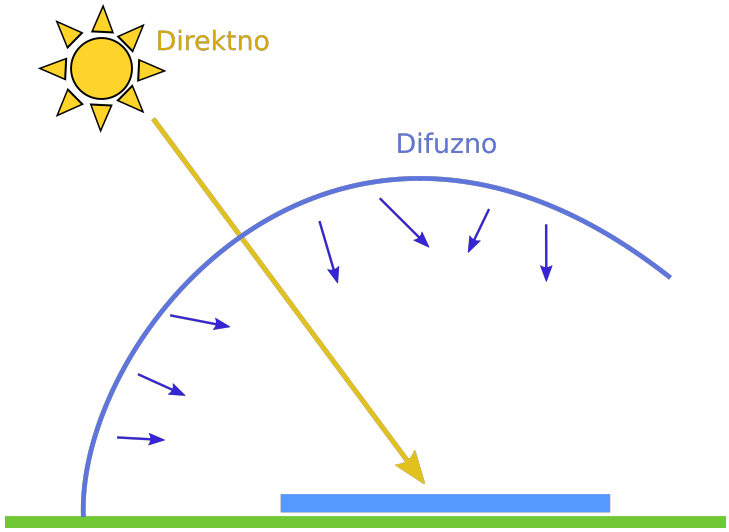
U ovom poglavlju bavićemo se jednim vrlo praktičnim problemom, koji je autoru i saradnicima i poslužio kao jedan od osnovnih motiva za rad sa NMPFZ. Naime, radi se o „večitom jazu" između rezultata koje daje model i stvarnih merenja. Kako ovaj jaz smanjiti, tj. kako pomiriti model i stvarne podatke? Primera radi, solarne elektrane su prilično dobro pokrivene različitim fizičkim modelima koji daju odlične rezultate, ali **samo ako su svi ulazni parametri i parametri samog sistema poznati**, što uglavnom nije slučaj. Prvo, podaci o vremenu kao što su temperatura, komponente sunčevog zračenja i brzina vetra nisu sasvim precizni. Tu je i faktor starenja samih panela koji umanjuje njihovu efikasnost, moguća prekrivenost snegom i prašinom i razni drugi faktori koji realno utiču na proizvodnju.

Sa druge strane, bilo koji solarni sistem novije proizvodnje nudi instant merenje izlazne snage naizmenične struje na izlazu iz invertera. Pokušaćemo da odgovorimo na pitanje da li je moguće ove merene podatke „vratiti" u model, tj. njihovim korišćenjem poboljšati predviđanje modela u budućnosti. Dodatno treba naglasiti da model solarnog sistema koji ćemo demonstrirati Dobos [Dob14] i nije zasnovan na običnoj ili parcijalnoj diferencijalnoj jednačini, već na jednostavnom analitičkom izrazu. Strogo gledano, moguće je iskoristiti i običnu duboku neuronsku mrežu da opiše ponašanje takve jednačine i manuelno formirati i funkciju gubitka koja će ispoštovati i jednačinu i merene podatke. Međutim, NMPFZ biblioteka kao što je DeepXDE pruža dobar softverski okvir koji u velikoj meri olakšava rad sa ovakvim problemima.

U nastavku ćemo objasniti osnove na kojima funkcionišu solarne elektrane, uključujući značajne ulazne varijable, način funkcionisanja i karakteristike samih uređaja.

7.1.1 Komponente zračenja

Količina sunčeve energije prikupljena od strane solarnog panela između ostalog zavisi od **njegove orijentacije**. Na primer, solarni panel okrenut ka zapadu će prikupljati malu količinu sunčeve energije tokom jutra jer je orijentisan ka tamnijem delu neba. Sa pomeranjem sunca, solarni panel se sve više obasjava, pa sakuplja veće količine zračenja. Ovo je ključ *Plane of Array* (POA) koncepta koji podrazumeva količinu sunčeve svetlosti koja se može prikupiti za zadati položaj panela. Lokacija, nagnutost i orijentacija panela su od ključnog značaja za procenu količine energije koja na panel pada. Osunčanost se izražava u vatima po kvadratnom metru $\frac{W}{m^2}$.



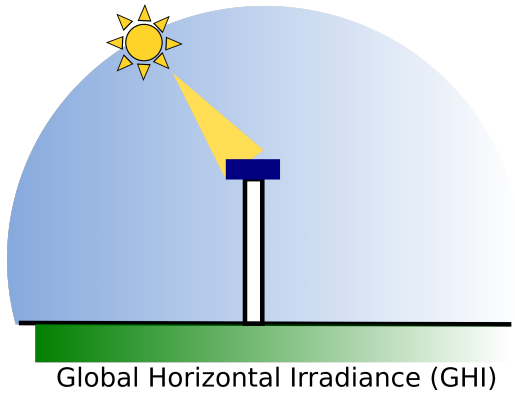
Sl. 7.1: Ilustracija direktnog i difuznog sunčevog zračenja

Kada se modeluje ozračenost ravni panela, iz praktičnih razloga, posmatraju se sledeće tri komponente:

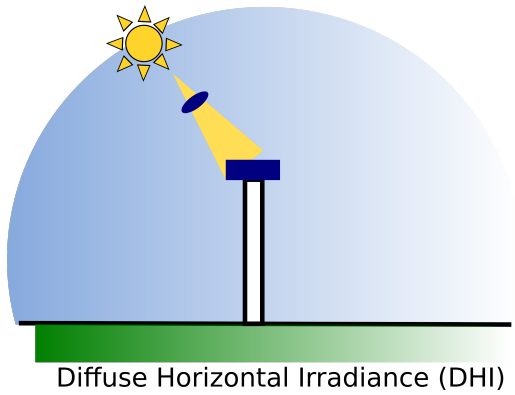
- `GHI` – **Globalna horizontalna ozračenost**, ukupna jačina sunčeve svetlosti koja pada na horizontalnu ravan, Sl. 7.2.
- `DHI` – **Difuzna horizontalna ozračenost**, deo sunčeve svetlosti koja pada na horizontalnu ravan, ali koja ne dolazi direktno od sunca, Sl. 7.3.

- `DNI` – **Direktna normalna ozračenost**, deo sunčeve svetlosti koja dolazi direktno od sunca, Sl. 7.4.

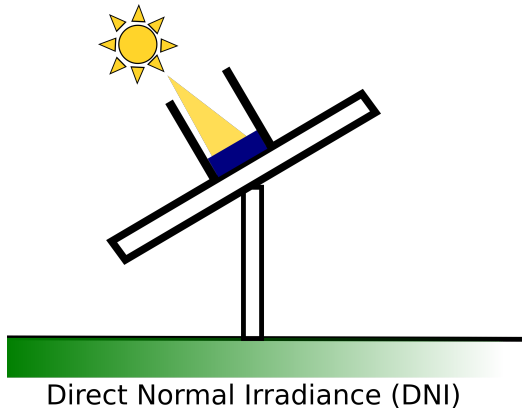
Svaka od navedenih komponenti se odgovarajućim mernim postupkom i instrumentom posebno meri. Na primer, izračunavanje komponente direktnog zračenja `DNI` koja pada na panel rešava se jednostavno na osnovu upadnog ugla. Pronalaženje komponente difuzne ozračenosti `DHI` je složenije i može varirati u zavisnosti od atmosferskih uslova. U upotrebi su različiti pristupi za konverziju `DHI` u difuznu komponentu. Treća komponenta zračenja `GHI` je svetlost koja se odbija od tla pre nego što je sakupi foto-naponska ploča. Na Internetu se mogu pronaći besplatne baze istorijskih podataka. Na pojedinim plaćenim servisima mogu se naći i prognoze sve tri komponente. Serije na ovim servisima su uglavnom satne učestanosti.



Sl. 7.2: `GHI`- Globalna horizontalna ozračenost



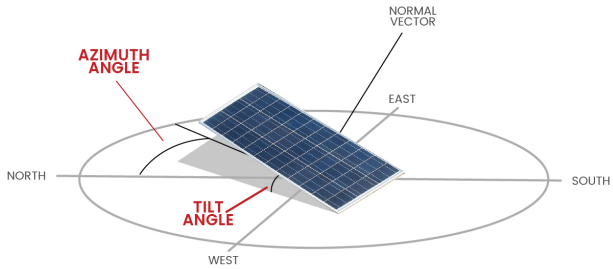
Sl. 7.3: `DHI` – Difuzna horizontalna ozračenost



Sl. 7.4: `DNI` – Direktna normalna ozračenost

7.1.2 Orijentacija i lokacija panela

Dva su ključna ugla koji definišu orijentaciju solarnog panela. Jedan određuje pravac postavljanja panela (sever, istog, jug, zapad), a drugi određuje nagnutost solarnog panela u odnosu na horizontalnu ravan. `Azimuth` određuje pravac postavljanja panela, pri čemu dogovorno važi da je sever 0, istok 90, jug 180, a zapad 270 stepeni. `Tilt` koji određuje nagnutost solarnog panela ima vrednost 0 ako je panel postavljen horizontalno, a vrednost 90 ukoliko je postavljen potpuno vertikalno. U zavisnosti od načina postavljanja sistema, i jedan i drugi ugao mogu biti fiksne vrednosti ili vremenske serije. Na takvim sistemima se obično `tilt` menja, prateći kretanje sunca na nebu i obezbeđujući veću ozračenost normalnom komponentom.



Sl. 7.5: Orijentacija solarnog panela

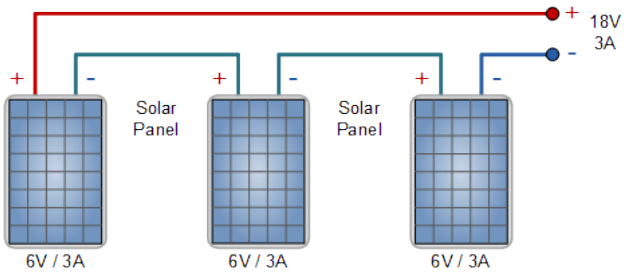
Određivanje tačne **lokacije panela** je veoma važno prilikom računanja ugla koji Sunce zaklapa sa panelom u različito doba godine. Parametri za računanje ugla koji sunce

zaklapa sa panelom su:

- geografska širina,
- geografska dužina,
- nadmorska visina i
- vremenska zona.

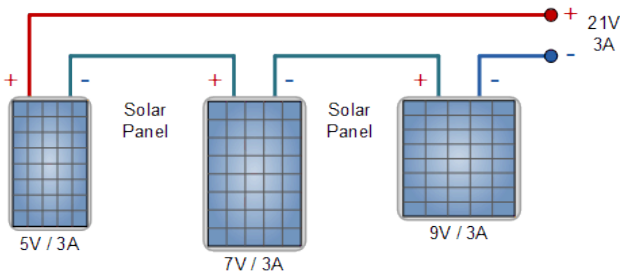
7.1.3 Načini povezivanja panela

Solarni paneli se u električno kolo mogu redno (serijski) i paralelno. Redna veza sumira napon u električnom kolu, dok paralelna veza povećava jačinu struje. Moguće je napraviti i kombinaciju redne i paralelne veze.

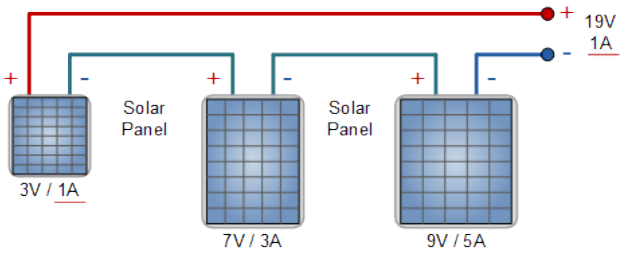


Sl. 7.6: **Serijska veza solarnih panela istih karakteristika**. Svi solarni paneli su istog tipa i imaju istu izlaznu snagu. Ukupan napon električnog kola je zbir napona na svakom panelu. U ovom primeru imamo 3 panela koji proizvode napon od 6V i struju jačine 3A, odnosno električno kolo ima napon od 18V i struju jačine 3A. Ukupna snaga veze je $18V \cdot 3A = 54W$ pri maksimalnoj osunčanosti[18].

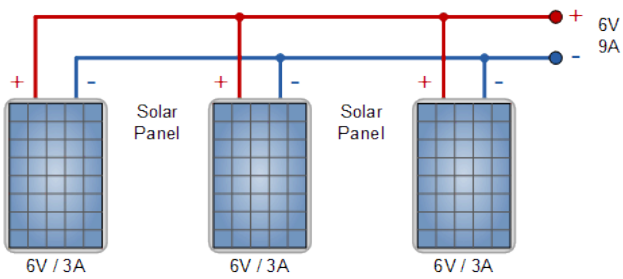
[18] https://www.alternative-energy-tutorials.com/solar-power/connecting-solar-panels-together.html



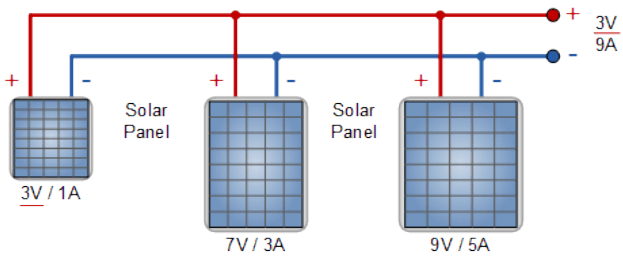
Sl. 7.7: **Serijska veza solarnih panela različitih voltaža**. U ovom primeru svi solarni paneli su različitih tipova, imaju različitu snagu, ali im je zajednička maksimalna jačina struje. Kada su vezani serijski, zajedno proizvode električni napon od 21V i struju jačine 3A, odnosno snaga je 63W. Jačina struje je ista kao i u prethodnom primeru, ali je promenjen napon (5V+7V+9V).



Sl. 7.8: **Serijska veza panela različitih napona i jačina električne struje**. U ovoj metodi solarni paneli su različitih tipova, svaki panel ima različit napon, jačinu struje i snagu. Ukupni napon električnog kola ponovo se računa kao zbir napona na svakom solarnom panelu (3V+7V+9V), dok je jačina struja u kolu ograničena panelom s najmanjom jačinom struje - 1A. Time je snaga limitirana na samo 19W od mogućih 69W. Upotreba solarnih panela različitih struja nije efikasna u rednoj vezi.



Sl. 7.9: **Paralelna veza solarnih panela istih karakteristika**. Svi solarni paneli na slici imaju iste karakteristike, napon, jačinu električne struje i snagu. Napon na svakom panelu je 6V pa je i ukupan napon kola 6V. Jačina struje na izlazu predstavlja zbir svih jačina električne struje na panelima 3A+3A+3A=9A. Ostvarena snaga pri potpunoj osunčanosti panela iznosi 54W.



Sl. 7.10: **Paralelna veza solarnih panela različitih napona i jačina struje**. Da bi radili u paralelnoj vezi, svi solarni paneli moraju imati isti napon, odnosno napon na svim panelima biće jednak najmanjem naponu na jednom od panela. Dakle, ukupan napon kola iznosi 3V, dok je ukupna jačina električne struje određena zbirom struja 1A+3A+5A=9A. Snaga iznosi samo 27W. Zbog ovih gubitaka ne preporučuje se paralelna veza solarnih panela različitih napona.

7.1.4 Nominalna snaga panela (*Peak Power - kWp*)

Merenja snage solarnih panela u laboratoriji ili fabrici vrši se pod standardizovanim uslovima. Ti uslovi definisani su integracionim standardnom IEC-60904-1 i to su:

- Intenzitet osunčanosti iznosi $1000W/m^2$ na celoj površini solarnog panela. U realnim uslovima ova vrednost je neretko veća.
- Temperatura panela iznosi 25°C. Spektar svetlosti mora biti isti kao globalni spektar svetlosti definisan u IEC 60904-3. Odgovara spektru svetlosti po sunčanom danu

sa položajem sunca oko 40° iznad horizonta i panelom koji je okrenut prema suncu, a sa horizontom zaklapa ugao od 40°.

Izmerena snaga pri ovim uslovima naziva se nominalna snaga ili snaga u piku – *Peak Power*. Nominalna snaga izražava se u kilovat-piku *kWp*. Ako nije poznata ukupna deklarisana nominalna snaga solarnih panela, a poznati su površina solarnih panela m^2 i deklarisana efikasnost u %, može se izračunati po formuli:

$$NominalnaSnaga = 1\frac{kW}{m^2} \cdot Povrsina \cdot \frac{Efikasnost}{100}$$

U većini slučajeva nominalna snaga je poznata i data je u specifikaciji proizvoda od strane proizvođača. Nominalna snaga još se naziva i maksimalna snaga i označava sa P_{max}.

7.1.5 Inverter

Solarnim panelima proizvodi se jednosmerna struja (DC). Da bi se upotrebila proizvedena električna energija, potrebno je izvršiti DC/AC konverziju. U upotrebi su različiti tipovi invertera:

- *Grid-tie inverter* - Priključeni su na distributivnu električnu mrežu, odnosno proizvedena električna energija prosleđuje se distributivnoj električnoj mreži. Za njihov rad nije potrebna baterija.
- *Off-grid inverter* – Poznati su i kao nezavisni inverteri. Konvertuju jednosmernu struju iz baterija u naizmeničnu. Uglavnom se koriste za više domaćinstva, ili stambenu zgradu.
- *Hybrid inverter* – Konvertuju DC u AC i mogu se koristiti i kao *off-grid* i kao *grid--tie* sistemi.
- *String inverter* – Najčešće se koriste u domaćinstvima. Nazivaju se „string” jer se na njih priključuje niz solarnih panela. Može se priključiti i više nizova odjednom.

Svaki inverter ima definisanu maksimalnu snagu. To je važno iz dva razloga: (1) elektronske komponente invertera dizajnirane su za rad sa određenim opsegom napona i (2) i sami solarni paneli su dizajnirani za rad do određene snage.

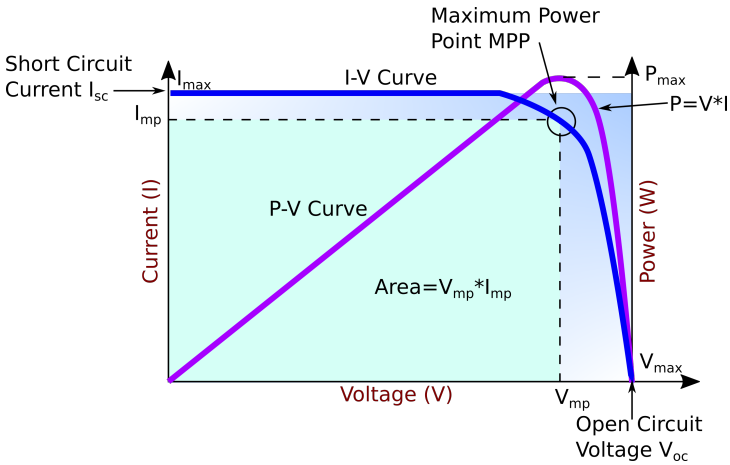
Inverteri, naravno, ne mogu pružiti veću izlaznu snagu od propisane. Kada se na ulaz invertera dovodi jednosmerna struja snage veće od propisane ulazne snage pojavljuje se odsecanje invertera, tj. inverter ima istu izlaznu snagu i pored povećanja snage na ulazu. Inverter može menjati napon na ulazu kako bi smanjio snagu na ulazu. Povećava operativni napon solarnih panela preko njihovog definisanog maksimuma, čime se smanjuje jačina proizvedene struje, odnosno umanjuje se snaga na ulazu invertera.

Razmotrimo situaciju gde su upareni solarni paneli snage 6 kW sa DC/AC inverterom snage 5 kW. Na prvi pogled izgleda da će dosta snage biti izgubljeno, ali veliki broj proizvođača preporučuju upravo ovakav odnos. Ovde uvodimo pojam odnosa snaga DC i AC poznat i kao *Inverter Load Ratio* - ILR. U ovom primeru taj odnos iznosi 1,2 (6kW/5kW). Projektanti ovakvih sistema su generalno konzervativni po pitanju DC/AC odnosa. Većina smatra da je odnos 1,1 idealan, a 1,2 prejak. Ipak, odnos 1,2 dovodi do najmanjih gubitaka, dok 1,25 ili 1,3 mogu ostvariti određene ekonomske benefite pri kupovini, jer se koriste jeftiniji inverteri[19].

7.1.6 I-V karakteristika solarne ćelije

Kriva I-V karakteristike prikazuje odnos struje i napona solarne ćelije, solarnog panela ili niza panela. Detaljno opisuje efikasnost konverzije solarne energije u električnu. Poznavanje električne I-V karakteristike i nominalne snage P_{max} panela je ključno za određivanje efikasnosti.

Intenzitet zračenja kojim se obasjava solarna ćelija određuje intenzitet struje, dok **povećanje temperature solarne ćelije smanjuje napon**. Kriva I-V karakteristike je grafička reprezentacija operacija u solarnoj ćeliji ili panelu sumirajući odnos između struje i napona, odnosno osunčanosti i temperature. Kriva pruža potrebne informacije za konfiguraciju solarnog panela, kako bi se konfigurisao za rad blizu svoje optimalne snage (*Peak Power*).



Sl. 7.11: Karakteristična I-V kriva

[19] https://help.helioscope.com/article/248-understanding-dc-ac-ratio

Na Sl. 7.11 vidi se I-V karakteristika (plava linija) tipične silikonske solarne ćelije pri normalnim uslovima. Snaga isporučena od strane jedne solarne ćelije ili panela je proizvod izlazne struje i napona. Kriva snage u zavisnosti od napona označena je ljubičastom bojom. Solarna ćelija je **pasivni uređaj u električnom kolu**. I-V kriva prikazuje sve mogućnosti za rad solarne ćelije, ali stvarni odnos struje i napona zavisiće od dodatnog opterećenja u električnom kolu.

Na primer, kada je na ćeliju priključena baterija, napon je 12V, a struja je visoka. S druge strane, kada je priključen potrošač, menja se odnos struje i napona. Razmotrimo dva krajnja slučaja:

- **Otvoreno kolo** nije povezano na opterećenje. Tada je struja na nuli, a napon ima maksimalnu vrednost. Takav napon naziva se naponom otvorenog kola, *Open Circuit Voltage*, odnosno V_{OC}.
- **Kolo kratkog spoja**, kada su pozitivni i negativni kraj solarne ćelije u kratkom spoju. Napon na solarnoj ćeliji je jednak nuli, dok je struja maksimalna. Takva struja naziva se strujom kratkog spoja - *Short Circuit Current*, I_{SC}.

Za nas je najzanimljiviji slučaj u kojem kombinacija struje i napona daje najveću vrednost snage. Označimo te vrednosti sa I_{MP} i V_{MP}. To je tačka u kojoj solarna ćelija generiše maksimum snage i označena je u gornjem desnom uglu zelenog pravougaonika na slici oznakom MPP (*Maximum Power Point*). Dakle, idealna proizvodnja solarne ćelije definisana je tačkom *MPP* koja se nalazi na prevoju I-V karakteristične krive. Odgovarajuće vrednosti za I_{MP} i V_{MP} mogu se proceniti na osnovu napona otvorenog kola $V_{MP} \approx (0,85 - 0,9) \cdot V_{OC}$ i struje kratkog spoja $V_{MP} \approx (0,85 - 0,95) \cdot I_{SC}$.

S obzirom da na napon solarnog panela **značajno utiče njegova temperatura**, stvarne vrednosti izlazne snage mogu varirati. Do sada smo razmatrali I-V karakterističnu krivu jedne solarne ćelije ili jednog panela. Kada imamo više uvezanih solarnih panela kriva I-V karakteristike ima isti oblik, samo su vrednosti skalirane. Kao što smo već pomenuli, paneli mogu biti uvezani serijski ili paralelno, odnosno mogu proizvesti veći napon ili veću struju. U svakom slučaju, gornji desni ugao osenčenog pravougaonika označavaće MPP. Postoje još dva značajna parametara koji opisuju rad panela:

- FF (*Fill Factor*) - Prestavlja odnos maksimalne snage koju niz solarnih panela može da proizvede pod normalnim uslovima i proizvoda struje kratkog spoja i napona otvorenog kola, $FF = P_{max}/(I_{SC} \cdot U_{OP})$. Što je vrednost bliža jedinici, može se proizvesti više snage. Uobičajene vrednosti su između 0,7 i 0,8.
- %Eff (*Percent Efficiency*) - Efikasnost niza solarnih panela je odnos između maksimalne električne snage koju mogu proizvesti i osunčanosti. Danas je efikasnost panela uglavnom oko 10% do 12%, u zavisnosti od tipa tehnologije.

U narednom odeljku *Osnovne jednačine modela* (stranica 113) izložićemo osnovne jednačine koje ćemo koristiti za formiranje modela baziranog na NMPFZ.

7.2 Osnovne jednačine modela

Kao što je već najavljeno u prethodnom odeljku *Uvod* (stranica 103), za modelovanje metodom NMPFZ koristićemo jednostavan *PVWAtts* model Dobos [Dob14]. Vertikala modelovanja sastoji se iz tri koraka:

- proračun temperature solarnog panela,
- proračun izlazne snage jednosmerne struje (*PVWatts* u užem smislu) i
- proračun izlazne naizmenične struje na izlazu iz invertera.

7.2.1 Proračun temperature solarnog panela

Zagrevanje solarnog panela utiče na promenu napona, pa je proračun temperature panela značajna stavka u modelovanju performansi. U širokoj upotrebi su dva modela, i to SAPM (*Sandia Array Performance Model*) i PVSyst. Mi ćemo koristiti ovaj prvi. SAPM metodologija modeliranja temperature ćelije T_{cell} data je sledećim parom jednačina:

$$\begin{aligned} T_m &= E \cdot e^{a+bW_s} + T_a \\ T_{cell} &= T_m + \frac{E}{E_0}\Delta T, \end{aligned} \tag{7.1}$$

gde E predstavlja intenzitet *POA* zračenja, W_c predstavlja brzinu vetra na visini od 10 metara, T_a temperaturu vazduha, dok je E_0 referentna vrednosti intenziteta *POA* zračenja, za koju se obično uzima vrednost od $1000W/m^2$.

Vrednosti parametara $a, b, \Delta T$ su specifične za različite modele solarnih panela u zavisnosti od načina njihove izrade (*glass/glass* ili *glass/polymer*) i načina na koji se postavljaju (*open rack, close roof, open rack, insulated back*).

7.2.2 Izlazna snaga jednosmerne struje

Model *PVWatts* prvi put je objavljen od strane *National Renewable Energy Laboratory (NREL)*. Jednačine na kojima je baziran su relativno jednostavne. Ulazne varijable su:

- Efektivna osunčanost G_{poaeff} - Ukupna ozračenost u jedinicama W/m^2. Podrazumeva se da smo već uzeli u obzir gubitke na osnovu ugla pod kojim je panel postavljen, ali ne i druge gubitke, npr. zbog prašine, spektralne gubitke, itd.
- Temperatura panela T_c koja se dobija iz jednačine (7.1).

Izraz glasi:

$$P_{dc} = \frac{G_{poaeff}}{1000} P_{dc0}(1 + \gamma_{pdc}(T_{cell} - T_{ref})), \tag{7.2}$$

gde su P_{dc0} i T_{ref} nominalna snaga panela i referentna temperatura (podrazumevano 25° C), respektivno. γ_{pdc} predstavlja temperaturni koeficijent u jedinicama $1/C$ sa tipičnim vrednostima u intervalu od -0,002 do -0,005 po stepenu Celzijusa. Što je ovaj koeficijent viši po apsolutnoj vrednosti, to proizvodnja više opada zagrevanjem panela.

7.2.3 *PVWatts* model invertera

PVWatts model invertera Dobos [Dob14] računa njegovu efikasnost η kao funkciju ulazne snage jednosmerne struje:

$$\eta = \frac{\eta_{nom}}{\eta_{ref}}(-0,0162\zeta - \frac{0,0059}{\zeta} + 0,9858), \tag{7.3}$$

gde je $\zeta = P_{dc}/P_{dc0}$ i $P_{dc0} = P_{ac0}/\eta_{nom}$.

Tada je snaga izlazne naizmenične struje:

$$P_{ac} = \min(\eta P_{dc}, P_{ac0}).$$

Veličina P_{dc} predstavlja ulaznu snagu jednosmerne struje i data je u istim jedinicama kao P_{dc0}. P_{dc0} je gornja granica ulazne snage. Koeficijenti η_{nom} i η_{ref} imaju podrazumevane vrednosti od 0,96 i 0,9637 respektivno.

7.3 Primer proračuna proizvodnje

7.3.1 Analitički model

U ovom odeljku ćemo upotrebiti metodologiju prikazanu u *Uvod* (stranica 103) i jednostavni model *PVWatts* objašnjen u *Osnovne jednačine modela* (stranica 113) upotrebiti na jednom realnom primeru. U pitanju je solarna elektrana u Centralnoj Srbiji sa 146 instalisanih panela na krovu jedne zgrade. Sve karakteristike instalacije su nam poznate (ugao nagiba, azimut, vrsta panela, karakteristike invertora i slično). Od podataka imamo i serije temperature vazduha i komponenti osunčanosti, tako da lako možemo analitički, po jednačinama (7.1), (7.2) i (7.3) da izračunamo proizvodnju. U slučaju da posedujemo vremensku prognozu, možemo približno i da predvidimo buduću proizvodnju koristeći standardni *PVWatts* model.

Iako matematika nije komplikovana, za programsku implementaciju ovog analitičkog modela koristićemo pomoć biblioteke PVLIB Holmgren *et al.* [HHM18]. Na narednom listingu prikazan je deo koda koji se tiče učitavanja, pripreme podataka i izvođenja *PVWatts* modela.

Listing 7.1: Implementacija *PVWatts* (analitičkog) modela proizvodnje

```
1  import numpy as np
2  import pandas as pd
3  import pvlib
4  from pvlib.location import Location
5
6  # Ucitaj podatke o vremenu i proizvodnji
7  solcast_data = pd.read_csv("data_21.08.16_22.10.14.csv", index_col=
   ↪"Time")
8  solcast_data.index = pd.to_datetime(solcast_data.index, dayfirst=True)
9
10 # Nagib, azimut i lokacija panela
11 surface_tilt = 7
12 surface_azimuth = 290
13 location = Location(latitude=43.905410, longitude=20.341986,␣
   ↪altitude=243, tz="Europe/Belgrade", name="Pons Cacak")
14
15 # Pomeri vreme za pola sata, izracunaj poziciju sunca u svakom trenutku
16 times = solcast_data.index - pd.Timedelta('30min')
17 solar_position = location.get_solarposition(times)
18 solar_position.index += pd.Timedelta('30min')
19
20 # Novi dataframe sa vrednostima za vrednosti osuncanosti DNI, GHI, DHI
```

(continues on next page)

(nastavak sa prethodne stranice)

```
df_poa = pvlib.irradiance.get_total_irradiance(
    surface_tilt=surface_tilt,
    surface_azimuth=surface_azimuth,
    dni=solcast_data['DNI'],
    ghi=solcast_data['GHI'],
    dhi=solcast_data['DHI'],
    solar_zenith=solar_position['apparent_zenith'],
    solar_azimuth=solar_position['azimuth'],
    model='isotropic')

# Izvuci ukupnu POA vrednost iz df_poa
E_data = df_poa["poa_global"]
# Izvuci temperaturu vazduha
T_data = solcast_data["Tamb"]
# Izvuci proizvodnju P
P_data = solcast_data["P"]

# Nedelju dana za treniranje
E_data_train = E_data.loc["2021-10-31":"2021-11-06"]
T_data_train = T_data.loc["2021-10-31":"2021-11-06"]
P_data_train = P_data.loc["2021-10-31":"2021-11-06"]

# Sledecih nedelju dana za testiranje
E_data_test = E_data.loc["2021-11-07":"2021-11-13"]
T_data_test = T_data.loc["2021-11-07":"2021-11-13"]
P_data_test = P_data.loc["2021-11-07":"2021-11-13"]

#
# Parametri modela
#
pdc0 = 0.375 # nominal power [kWh]
Tref = 25.0 # cell reference temperature
gamma_pdc = -0.005 # influence of the cell temperature on PV system
pdc0_inv = 50
eta_inv_nom = 0.96
eta_inv_ref = 0.9637
pac0_inv = eta_inv_nom * pdc0_inv # maximum inverter capacity
a = -2.98 # cell temperature parameter
b = -0.0471 # Wind coefficient
E0 = 1000 # reference irradiance
deltaT = 1 # cell temperature parameter
num_of_panels = 146 # Broj panela u instalaciji

#
# Originalni PVWAtts model
#
```

(continues on next page)

(nastavak sa prethodne stranice)

```
67 def orig_pvwatts_model(x):
68     Ta = x[:,0:1] # Temperatura vazduha
69     E = x[:,1:2] # Ukupna POA osuncanost
70 
71     Tm = E * np.exp(a+b*2) + Ta # Brzina vetra uzeta kao prosecna od 2
   ↪m/s
72     Tc = Tm + E/E0*deltaT
73     P_dc_temp = ((Tc-Tref) * gamma_pdc + 1.)
74     P_dc = (E * 1.e-03 * pdc0 * P_dc_temp) * num_of_panels
75     zeta = (P_dc+1.e-2)/pdc0_inv
76 
77     eta = eta_inv_nom/eta_inv_ref * (-0.0162*zeta - 0.0059/zeta + 0.
   ↪9858)
78     eta[eta<0] = 0.
79     ac = np.minimum(eta*P_dc, pac0_inv)
80 
81     return ac
```

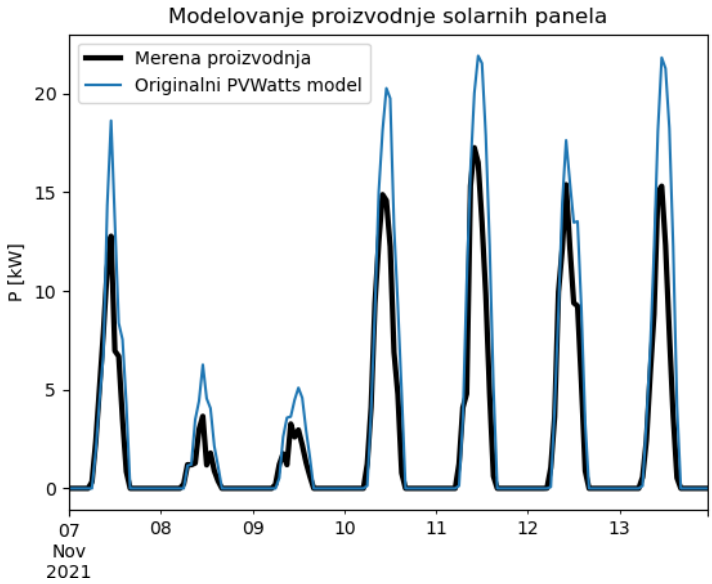
Nakon standardnih importa biblioteka, učitavamo satne serije podataka o vremenu:

- `Ta` - temperatura vazduha,
- `DNI`, `GHI`, `DHI` - komponente osunčanosti

i podatke o proizvodnji `P` u *Pandas* okvir. Na žalost, nemamo dostupne podatke o brzini vetra na lokaciji, pa ćemo tu komponentu uzeti kao prosečnu na tom podneblju, tj. vrednost od $2\,m/s$. Funkcija `pvlib.irradiance.get_total_irradiance()` biblioteke *PVLIB* nam izračunava položaj sunca za bilo koji vremenski trenutak i bilo koju lokaciju na Zemlji, a na osnovu njega i ukupnu osunčanost panela čiji je položaj dat uglovima `surface_tilt` i `surface_azimuth`. Kao ulaz, ova metoda uzima komponente `DNI`, `GHI` i `DHI` definisane u odeljku *Komponente zračenja* (stranica 104).

Nakon ekstrakcije perioda od nedelju dana od 31. oktobra do 6. novembra i drugog perioda od 7. do 13. novembra, postavljamo sve parametre koji se koriste u jednačinama (7.1), (7.2) i (7.3), onako kako najbolje odgovara samoj instalaciji. Svi parametri osim tri su konstante prihvaćene u literaturi. Ta tri parametra čija vrednost može da se podešava su `a` i `b` iz izraza (7.1) i `gamma_pdc` iz izraza (7.2). Njihove vrednosti (-2,98, -0.0471 i -0,005 respektivno) zadajemo prema vrsti panela i načinu postavljanja instalacije.

Poređenje rezultata dobijenih čistim modelom i merenih vrednosti može se videti na Sl. 7.12. Očigledno je da postoji značajno odstupanje, tj. model daje više vrednosti od merenja. Koren srednje kvadratne greške (*RMSE*) iznosi čak 2,46 kW. Određeni doprinos ovako visokoj vrednosti greške sigurno je posledica činjenice da hlađenje panela usled uticaja vetra nismo uzeli u obzir usled nedostatka podataka o vetru.



Sl. 7.12: Poređenje čistog *PVWAtts* modela sa merenom proizvodnjom

Nijedan model nije savršena slika stvarnosti i ne može da obuhvati sve faktore koji utiču na proizvodnju. Čak iako je model savršen (a ne može biti), moguće su pojave grešaka pri merenju temperature, osunčanosti, brzine vetra. Dalje, kod samog modela imamo više parametara čije vrednosti uzimamo iz literature i deklaracije proizvođača. Realne vrednosti tih parametara sigurno odstupaju od tih vrednosti i menjaju se tokom životnog veka uređaja.

7.3.2 Podešavanje parametra pomoću NMPFZ

Postavlja se pitanje da li vrednosti koje daje model mogu biti približnije realnim vrednostima samo podešavanjem parametara modela. Probaćemo da iskoristimo činjenicu da podatke o proizvodnji za nedelju od 31. oktobra imamo i da pomoću NMPFZ probamo da „pomirimo” izlaz analitičkog modela i merenja proizvodnje tako što ćemo parametar `a` jednačine (7.2) proglasiti nepoznatim. Osnovna ideja je da se NMPFZ trenira i jednačinom i podacima i da kao izlaz isporuči i model sa manjom greškom i novu, bolju vrednost parametra `a`.

Na sledećem listingu mogu se videti interesantni delovi implementacije ove ideje:

Listing 7.2: Inverzni problem podešavanja parametara PVWatts/SAPM modela

```
# Parametar "a" pustamo da se trenira
a_var = dde.Variable(-4.0)

#
# Jednacina koju koristi PINN
#
def pvwatts_eq(x, y):
    Ta = x[:,0:1] # Temperatura vazduha
    E = x[:,1:2] # Ukupna POA osuncanost

    Tm = E * tf.exp(a_var+b*2) + Ta # Brzina vetra uzeta kao prosecna od 2 m/s
    Tc = Tm + E/E0*deltaT
    P_dc_temp = ((Tc-Tref) * gamma_pdc + 1)
    P_dc = (E * 1.e-03 * pdc0 * P_dc_temp) * num_of_panels
    zeta = (P_dc+1.e-2)/pdc0_inv

    eta = eta_inv_nom/eta_inv_ref * (-0.0162*zeta - 0.0059/zeta + 0.9858)
    eta = tf.maximum(0., tf.sign(eta)) * eta
    ac = tf.minimum(eta*P_dc, pac0_inv)

    return y - ac

# Imamo 168 tacaka sa merenjima proizvodnje. Pripremi strukturu za PointSet granicni uslov
train_points = np.zeros((168,2))
train_measured_production = np.zeros((168,1))
train_points[:,0] = T_data_train.to_numpy().T
train_points[:,1] = E_data_train.to_numpy().T
train_measured_production[:,0] =  P_data_train.to_numpy().T

# Imamo 168 tacaka sa merenjima za narednu nedelju za test
test_points = np.zeros((168,2))
test_measured_production = np.zeros((168,1))
test_points[:,0] = T_data_test.to_numpy().T
test_points[:,1] = E_data_test.to_numpy().T
test_measured_production[:,0] =  P_data_test.to_numpy().T

# Minimumi i maksimumi T i E za kreiranje geometrije problema
minT, maxT = min(train_points[:,0]), max(train_points[:,0])
minE, maxE = min(train_points[:,1]), max(train_points[:,1])

geom = dde.geometry.Rectangle([minT, minE], [maxT, maxE])
bc_y = dde.icbc.PointSetBC(train_points, train_measured_production,
```

(continues on next page)

(nastavak sa prethodne stranice)

```
    ↪component=0)
43
44  # Isti broj kolokacionih tacaka za jednacinu i za granicne uslove.␣
    ↪Moze i drugacije.
45  data = dde.data.PDE(geom, pvwatts_eq, [bc_y], 168, 168, solution =␣
    ↪orig_pvwatts_model, num_test=100)
46
47  layer_size = [2] + [30] * 5 + [1]
48  activation = "tanh"
49  initializer = "Glorot uniform"
50  net = dde.nn.FNN(layer_size, activation, initializer)
51
52  variable_a = dde.callbacks.VariableValue(a_var, period=1000)
53  model = dde.Model(data, net)
54
55  model.compile(optimizer="adam", lr=0.001, metrics=["l2 relative error
    ↪"], external_trainable_variables=[a_var])
56  losshistory, train_state = model.train(iterations=20000,␣
    ↪callbacks=[variable_a])
57  predicted_test = model.predict(test_points)
58
59  # Predikcije manje od nule nemaju smisla. Nuluj ih.
60  predicted_test[predicted_test<0]=0
```

Kao i kod ranije obrađenih inverznih problema koji koriste biblioteku *DeepXDE*, postavljamo parametar kao varijablu čija se vrednost dobija procesom obučavanja:

```
a_var = dde.Variable(-4.0)
```

Ako pogledamo funkciju `pvwatts_eq(x, y)`, ona je gotovo identična funkciji `orig_pvwatts_model(x)` sa Listing 7.1. Razlika je u tome što `pvwatts_eq(x, y)` ne vraća vrednost snage, već funkciju gubitka, kao što smo već navikli. Još jedna razlika ogleda se u korišćenju *TensorFlow* logike umesto uslovnog izraza:

```
eta = tf.maximum(0., tf.sign(eta)) * eta
```

Ovaj izraz nije ništa drugo nego uslov da ako imamo nefizičku vrednost `eta<0` postavimo da je `eta=0`. Ovom formulacijom izbegavamo instrukciju uslovnog skoka, koja se na grafičkom procesoru izvodi dosta sporije od čistog računa u pokretnom zarezu.

U daljem toku programa treba da postavimo strukturu za poseban granični uslov `PointSet`, koji smo već koristili u odeljku *Inverzni problem* (stranica 47) i *Propagacija talasa u otvorenom kanalu* (stranica 62):

```
train_points = np.zeros((168,2))
train_measured_production = np.zeros((168,1))
train_points[:,0] = T_data_train.to_numpy().T
train_points[:,1] = E_data_train.to_numpy().T
train_measured_production[:,0] =  P_data_train.to_numpy().T

bc_y = dde.icbc.PointSetBC(train_points, train_measured_production,␣
↪component=0)
```

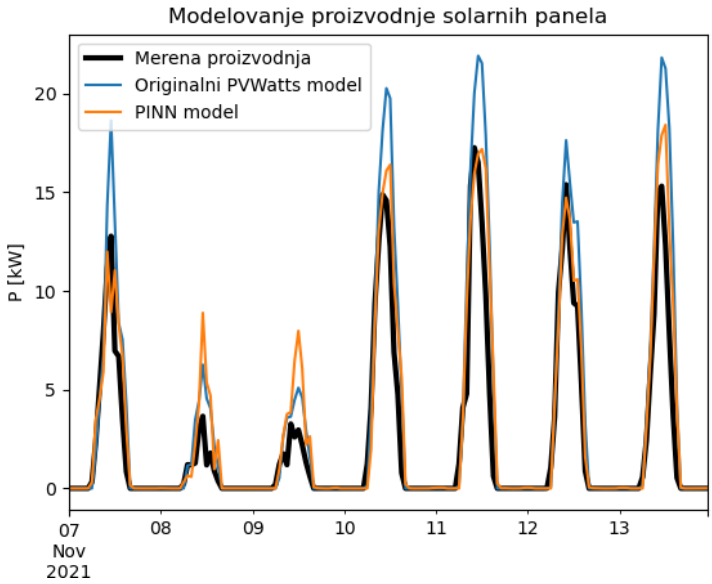
Imamo ukupno 168 tačaka (7x24) merenja proizvodnje koje ćemo iskoristiti u pokušaju da „spustimo" vrednosti koje daje originalni model. Geometriju problema koji rešavamo definišemo opsegom ulaznih varijabli temperature i ukupne osunčanosti, koje se uzimaju iz tabele podataka. Onda možemo da postavimo i kolokacione tačke na domenu naredbom:

```
data = dde.data.PDE(geom, pvwatts_eq, [bc_y], 168, 168, solution =␣
↪orig_pvwatts_model, num_test=100)
```

Za broj slučajnih kolokacionih tačaka unutar domena uzeli smo isti broj tačaka koliko imamo u `PointSet` graničnom uslovu. Nije nužno da broj tačaka bude jednak, pa ostavljamo čitaocu da eksperimentiše različitim vrednostima. Ostatak koda je manje-više isti kao kod svih drugih primera koji koriste DeepXDE za rešavanje inverznih problema. Tu je postavljanje arhitekture NMPFZ i hiper-parametara, algoritma optimizacije, stope obuke, *callback* funkcije za štampu trenutne vrednosti `a_var` tokom obuke i slično. Na kraju se anuliraju negativne vrednosti predviđene proizvodnje jer nemaju fizičkog smisla.

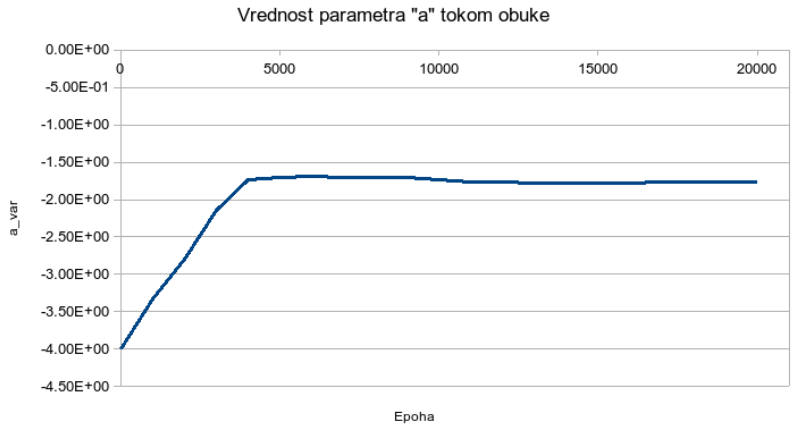
7.4 Rezultati

Na dijagramu Sl. 7.13 može videti rezultat pristupa opisanog u odeljku *Primer proračuna proizvodnje* (stranica 115). Radi poređenja, na slici je ostao prikazan i grafik sa Sl. 7.12.



Sl. 7.13: Poređenje NMPFZ modela i čistog *PVWAtts* modela sa merenom proizvodnjom

Očigledno je da NMPFZ model daje bolje rezultate od čistog *PVWatts* modela. To potvrđuje i greška koja je spuštena sa 2,51kW na 1,92kW. Ilustracije radi, dajemo i vrednost parametra `a_var` tokom obuke na Sl. 7.14. Umesto vrednosti iz literature -2,98, ispostavlja se da podacima više odgovara vrednost od oko -1,78.



Sl. 7.14: Vrednost parametra a tokom obuke

Sigurno je da model i dalje može da se podešava, recimo uključivanjem varijacije još nekog parametra, kao što je `gamma_pdc`. Međutim, ovde se time nećemo baviti jer smo u dovoljnoj meri postigli cilj, tj. pokazali način rada sa ovom vrstom problema. Čitalac može da proba i neki drugi pristup jer su svi potrebni podaci dostupni u repozitorijumu praktikuma.

Bibliografija

[Bea12] Jacob Bear. *Hydraulics of groundwater*. Courier Corporation, 2012.

[CHB20] Jayashree Chadalawada, HMVV Herath, and Vladan Babovic. Hydrologically informed machine learning for rainfall-runoff modeling: a genetic programming-based toolkit for automatic model induction. *Water Resources Research*, 56(4):e2019WR026933, 2020.

[Dob14] Aron P Dobos. Pvwatts version 5 manual. Technical Report, National Renewable Energy Lab.(NREL), Golden, CO (United States), 2014.

[DCK+13] Andrew Duncan, Albert S Chen, Edward Keedwell, Slobodan Djordjevic, and Dragan Savic. Rapids: early warning system for urban flooding and water quality hazards. 2013.

[GarzonKLT22] Alexander Garzón, Zoran Kapelan, J Langeveld, and Riccardo Taormina. Machine learning-based surrogate modelling for urban water networks: review and future research directions. *Water Resources Research*, pages e2021WR031808, 2022.

[HJ21] Ehsan Haghighat and Ruben Juanes. Sciann: a keras/tensorflow wrapper for scientific computations and physics-informed deep learning using artificial neural networks. *Computer Methods in Applied Mechanics and Engineering*, 373:113552, 2021.

[HHM18] William F Holmgren, Clifford W Hansen, and Mark A Mikofski. Pvlib python: a python package for modeling solar energy systems. *Journal of Open Source Software*, 3(29):884, 2018.

[Hux57] Andrew F Huxley. Muscle structure and theories of contraction. *Prog. Biophys. Biophys. Chem*, 7:255–318, 1957.

[Ihl98] Frank Ihlenburg. *Finite element analysis of acoustic scattering*. Springer, 1998.

[IS22] Milos Ivanovic and Visnja Simic. Efficient evolutionary optimization using predictive auto-scaling in containerized environment. *Applied Soft Computing*, 129:109610, 2022.

[ISS+15] Milos Ivanovic, Visnja Simic, Boban Stojanovic, Ana Kaplarevic-Malisic, and Branko Marovic. Elastic grid resource provisioning with wobingo: a parallel framework for genetic algorithm based optimization. *Future Generation Computer Systems*, 42:44–54, 2015.

[ISS17] Milos Ivanovic, Marina Svicevic, and Svetislav Savovic. Numerical solution of stefan problem with variable space grid method based on mixed finite element/finite difference approach. *International Journal of Numerical Methods for Heat and Fluid Flow*, 2017.

[KK19] Kian Katanforoosh and Daniel Kunin. Initializing neural networks. *DeepLearning. ai*, 2019.

[Kojic98] Miloš Kojić. *Metod konačnih elemenata: Linearna analiza. I*. Mašinski fakultet, 1998.

[LLF98] I.E. Lagaris, A. Likas, and D.I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9:987–1000, 1998.

[LMMK21] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde: a deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.

[Mar21] Stefano Markidis. The old and the new: can physics-informed deep-learning replace traditional linear solvers? *Frontiers in big Data*, pages 92, 2021.

[PU92] Dimitris C. Psichogios and Lyle H. Ungar. A hybrid neural network-first principles approach to process modeling. *AIChE Journal*, 38:1499–1511, 1992.

[RPK19] Maziar Raissi, Paris Perdikaris, and George 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*, 378:686–707, 2019.

[Rec04] Gerald W Recktenwald. Finite-difference approximations to the heat equation. *Mechanical Engineering*, 2004.

[SC09] Svetislav Savovic and James Caldwell. Numerical solution of stefan problem with time-dependent boundary conditions by variable space grid method. *Thermal Science*, 13:165–174, 2009.

[SSI19] Visnja Simic, Boban Stojanovic, and Milos Ivanovic. Optimizing the performance of optimization in the cloud environment–an intelligent auto-scaling approach. *Future Generation Computer Systems*, 101:909–920, 2019.

[Svivcevic20] Marina Svičević. *Višeskalni računarski model mišića zasnovan na makromodelu konačnih elemenata i Hakslijevom mikromodelu*. PhD thesis, Универзитет у Крагујевцу, Природно-математички факултет, 2020.

[WYP22] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why pinns fail to train: a neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.

Izdavač
Prirodno-matematički fakultet Kragujevac
Radoja Domanovića 12
http://www.pmf.kg.ac.rs
Kragujevac

CIP – Katalogizacija u publikaciji
Narodna biblioteka Srbije, Beograd

004.032.26:[539:519.673(075.8)(076)

IVANOVIĆ, Miloš, 1978–
Neuronske mreže podržane fizičkim zakonima : praktikum / Miloš Ivanović. – Kragujevac : Prirodno-matematički fakultet, 2023
(Niš : Grafika Galeb). – II, 129 str. : ilustr. ; 25 cm

Tiraž 70. – Bibliografija: str. 127–129 i uz tekst.

ISBN 978-86-6009-096-8

a) Neuronske mreže podržane fizičkim zakonima -- Vežbe

COBISS.SR-ID 117042185
